

RIFT  
Internet-Draft  
Intended status: Standards Track  
Expires: 8 September 2022

J. Head, Ed.  
T. Przygienda  
W. Lin  
Juniper Networks  
7 March 2022

RIFT Auto-EVPN  
draft-ietf-rift-auto-evpn-02

## Abstract

This document specifies procedures that allow an EVPN overlay to be fully and automatically provisioned when using RIFT as underlay by leveraging RIFT's no-touch ZTP architecture.

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 September 2022.

## Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Requirements Language . . . . .	3
2. Design Considerations . . . . .	3
3. System ID . . . . .	4
4. Fabric ID . . . . .	4
5. Auto-EVPN Device Roles . . . . .	5
5.1. All Participating Nodes . . . . .	5
5.2. ToF Nodes as Route Reflectors . . . . .	5
5.2.1. Data Center Interconnect Gateway Functions . . . . .	6
5.3. Leaf Nodes . . . . .	6
6. Auto-EVPN Variable Derivation . . . . .	8
6.1. Auto-EVPN Version . . . . .	8
6.2. MAC-VRF ID . . . . .	8
6.3. Loopback Address . . . . .	8
6.3.1. Leaf Nodes as Gateways . . . . .	9
6.3.2. ToF Nodes as Route Reflectors . . . . .	9
6.3.2.1. Single Plane Route Reflector Election Procedures . . . . .	9
6.3.2.2. Multiplane Route Reflector Election Procedures . . . . .	11
6.4. Autonomous System Number . . . . .	11
6.5. Router ID . . . . .	11
6.6. Cluster ID . . . . .	11
6.7. Route Target . . . . .	12
6.8. Route Distinguisher . . . . .	12
6.9. EVPN MAC-VRF Services . . . . .	12
6.9.1. Untagged Traffic in Multiple Fabrics . . . . .	13
6.9.1.1. VLAN . . . . .	13
6.9.1.2. VNI . . . . .	13
6.9.1.3. MAC Address . . . . .	13
6.9.1.4. IPv6 IRB Gateway Address . . . . .	13
6.9.1.5. IPv4 IRB Gateway Address . . . . .	13
6.9.2. Tagged Traffic in Multiple Fabrics . . . . .	14
6.9.2.1. VLAN . . . . .	14
6.9.2.2. VNI . . . . .	14
6.9.2.3. MAC Address . . . . .	14
6.9.2.4. IPv6 IRB Gateway Address . . . . .	14
6.9.2.5. IPv4 IRB Gateway Address . . . . .	15
6.9.3. Tagged Traffic in a Single Fabric . . . . .	15
6.9.3.1. VLAN . . . . .	15
6.9.3.2. VNI . . . . .	15
6.9.3.3. MAC Address . . . . .	15
6.9.3.4. IPv6 IRB Gateway Address . . . . .	16
6.9.3.5. IPv4 IRB Gateway Address . . . . .	16
6.9.4. Traffic Routed to External Destinations . . . . .	16
6.9.4.1. Route Distinguisher . . . . .	16
6.9.4.2. Route Target . . . . .	16

7. Operational Considerations . . . . .	17
7.1. RIFT Underlay and Auto-EVPN Overlay . . . . .	17
7.2. Auto-EVPN Analytics . . . . .	20
7.2.1. Auto-EVPN Global Analytics Key Type . . . . .	21
7.2.2. Auto-EVPN MAC-VRF Key Type . . . . .	22
8. Acknowledgements . . . . .	23
9. Security Considerations . . . . .	23
10. References . . . . .	23
10.1. Normative References . . . . .	23
Appendix A. Thrift Models . . . . .	24
A.1. common.thrift . . . . .	24
A.2. encoding.thrift . . . . .	24
A.3. common_evpn.thrift . . . . .	25
A.4. auto_evpn_kv.thrift . . . . .	28
Appendix B. Auto-EVPN Variable Derivation . . . . .	30
B.1. Variable Derivation Functions . . . . .	30
B.2. Variable Derivation Results . . . . .	42
Authors' Addresses . . . . .	87

## 1. Introduction

RIFT is a protocol that focuses heavily on operational simplicity. [RIFT] natively supports Zero Touch Provisioning (ZTP) functionality that allows each node in an underlay network to automatically derive its place in the topology and configure itself accordingly when properly cabled. RIFT can also disseminate Key-Value information contained in Key-Value Topology Information Elements (KV-TIEs) [RIFT-KV]. These KV-TIEs can contain any information and therefore be used for any purpose. Leveraging RIFT to provision EVPN overlays without any need for configuration and leveraging KV capabilities to easily validate correct operation of such overlay without a single point of failure would provide significant benefit to operators in terms of simplicity and robustness of such a solution.

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. Design Considerations

EVPN supports various service models, this document defines a method for the VLAN-Aware service model defined in [RFC7432]. Other service models may be considered in future revisions of this document.

Each model has its own set of requirements for deployment. For example, a functional BGP overlay is necessary to exchange EVPN NLRI regardless of the service model. Furthermore, the requirements are made up of individual variables, such as each node's loopback address and AS number for the BGP session. Some of these variables may be coordinated across each node in a network, but are ultimately locally significant (e.g. route distinguishers). Similarly, calculation of some variables will be local only to each device. RIFT contains currently enough topology information in each node to calculate all those necessary variables automatically.

Once the EVPN overlay is configured and becomes operational, RIFT Key-Value TIEs can be used to distribute state information to allow for validation of basic operational correctness without the need for further tooling.

### 3. System ID

The 64-bit RIFT System ID that uniquely identifies a node as defined in RIFT [RIFT].

### 4. Fabric ID

RIFT operates on variants of Clos substrate which are commonly called an IP Fabric. Since EVPN VLANs can be either contained within one fabric or span them, Auto-EVPN introduces the concept of a Fabric ID into RIFT.

This section describes an optional extension to LIE packet schema in the form of a 16-bit Fabric ID that identifies a nodes membership within a particular fabric. Auto-EVPN capable nodes MUST support this extension but MAY not advertise it when not participating in Auto-EVPN. A non-present Fabric ID and value of 0 is reserved as ANY\_FABRIC and MUST NOT be used for any other purpose.

Fabric ID MUST be considered in existing adjacency FSM rules so nodes that support Auto-EVPN can interoperate with nodes that do not. The LIE validation is extended with following clause and if it is not met, miscabbling should be declared:

```
(if fabric_id is not advertised by either node OR
  if fabric_id is identical on both nodes)
  AND
  (if auto_evpn_version is not advertised by either node OR
    if auto_evpn_version is identical on both nodes)
```

The appendix (Appendix A) details necessary changes to the RIFT LIE and Node-TIE thrift schema.

## 5. Auto-EVPN Device Roles

Auto-EVPN requires that each node understand its given role within the scope of the EVPN implementation so each node derives the necessary variables and provides the necessary overlay configuration. For example, a leaf node performing VXLAN gateway functions does not need to derive its own Cluster ID or learn one from the route reflector that it peers with.

### 5.1. All Participating Nodes

Not all nodes have to participate in Auto-EVPN, however if a node does assume an Auto-EVPN role, it MUST derive the following variables:

**\*IPv6 Loopback Address\***

Unique IPv6 loopback address used in BGP sessions.

**\*Router ID\***

The BGP Router ID.

**\*Autonomous System Number\***

The ASN for IBGP sessions.

**\*Cluster ID\***

The Cluster ID for Top-of-Fabric IBGP route reflection.

### 5.2. ToF Nodes as Route Reflectors

This section defines an Auto-EVPN role whereby some Top-of-Fabric nodes act as EVPN route reflectors. It is expected that route reflectors would establish IBGP sessions with leaf nodes in the same fabric. The typical route reflector requirements do not change, however determining which specific values to use requires further consideration.

ToF nodes performing route reflector functionality MUST derive the following variables:

**\*IPv6 RR Loopback Address\***

The source address for IBGP sessions with leaf nodes in case ToF won election for one of the route reflectors in the fabric.

**\*IPv6 RR Acceptable Prefix Range\***

Range of addresses acceptable by the route reflector to form a IBGP session. This range covers ALL possible IPv6 Loopback Addresses derived by other Auto EVPN nodes in the current fabric and other Auto-EVPN RRs addresses.

**\*Cluster ID\***

The Cluster ID for Top-of-Fabric IBGP route reflection.

### 5.2.1. Data Center Interconnect Gateway Functions

Implementations that require connectivity beyond the EVPN/VXLAN boundary can leverage Data Center Interconnect Gateway functionality. This requires additional considerations to ensure the appropriate reachability is present.

First - new VRFs and accompanying variable derivation is required, this is described below.

Second - additional route reflector election considerations in order to ensure that route reflectors with DCI gateway functionality are preferred. This is described later in the document in Section 6.3.2.

If DCI functionality is desired, the Top-of-Fabric nodes MUST be capable of routing toward the correct leaf node when it receives traffic from an external destination. Therefore, it MUST be capable of deriving the following types of variables:

**\*Route Distinguisher\***

The route distinguisher corresponding to a IP-VRF's IP prefix routes that MUST uniquely identify each node.

**\*Route Target\***

The route target that corresponds to an IP-VRF's IP prefix routes.

**\*VNI\***

The VNI that corresponds to the Type-5 IP prefix routes within an IP-VRF.

### 5.3. Leaf Nodes

Leaf nodes derive their role from realizing they are at the bottom of the fabric, i.e. not having any southbound adjacencies. Alternately, a node can assume a leaf node if it has only southbound adjacencies to nodes with explicit LEAF\_LEVEL to allow for scenarios where RIFT leaves do NOT participate in Auto-EVPN.

Leaf nodes MUST derive the following variables:

**\*IPv6 RR Loopback Addresses\***

Addresses of the RRs present in the fabric. Those addresses are used to build BGP sessions to the RR.

**\*EVis\***

Leaf node derives all the necessary variables to instantiate EVIs with layer-2 and optionally layer-3 functionality.

If a leaf node is required to perform layer-2 VXLAN gateway functions, it MUST be capable of deriving the following types of variables:

**\*Route Distinguisher\***

The route distinguisher corresponding to a MAC-VRF that uniquely identifies each node.

**\*Route Target\***

The route target that corresponds to a MAC-VRF.

**\*MAC VRF Name\***

This is an optional variable to provide a common MAC VRF name across all leaves.

**\*Set of VLANs\***

Those are VLANs provisioned either within the fabric or allowing to stretch across fabrics.

For each VLAN derived in an EVI the following variables MUST be derived:

**\*VLAN\***

The VLAN ID.

**\*Name\***

This is an optional variable to provide a common VLAN name across all leaves.

**\*VNI\***

The VNI that corresponds to the VLAN ID. This will contribute to the EVPN Type-2 route.

**\*IRB\***

Optional variables of the IRB for the VLAN if the leaf performs layer-3 gateway function.

## 6. Auto-EVPN Variable Derivation

As previously mentioned, not all nodes are required to derive all variables in a given network (e.g. a transit spine node may not need to derive any or participate in Auto-EVPN). Additionally, all derived variables are derived from RIFT's FSM or ZTP mechanism so no additional flooding beside RIFT flooding is necessary for the functionality.

It is also important to mention that all variable derivation is in some way based on combinations of System ID, MAC-VRF ID, Fabric ID, EVI and VLAN and MUST comply precisely with calculation methods specified in the Auto-EVPN Variable Derivation section to allow interoperability between different implementations. All foundational code elements are also mentioned there.

### 6.1. Auto-EVPN Version

This section describes extensions to both the RIFT LIE packet and Node-TIE schemas in the form of a 16-bit value that identifies the Auto-EVPN Version. Auto-EVPN capable nodes MUST support this extension, but MAY choose not to advertise it in LIEs and Node-TIEs when Auto-EVPN is not being utilized.

This section also describes an extension to the Node Capabilities schema indicating that a node supports Auto-EVPN.

The appendix (Appendix A) details necessary changes to the RIFT LIE, Node-TIE, and Node Capabilities thrift schema.

### 6.2. MAC-VRF ID

This section describes a variable MAC-VRF ID that uniquely identifies an instance of EVPN instance (EVI) and is used in variable derivation procedures. Each EVPN EVI MUST be associated with a unique MAC-VRF ID, this document does not specify a method for making that association or ensuring that they are coordinated properly across fabric(s).

### 6.3. Loopback Address

First and foremost, RIFT does not advertise anything more specific than the fabric default route in the southbound direction by default. However, Auto-EVPN nodes MUST advertise specific loopback addresses southbound to all other Auto-EVPN nodes so to establish MP-BGP reachability correctly in all scenarios.



Auto-EVPN nodes MUST derive a ULA-scoped IPv6 loopback address to be used as both the IBGP source address, as well as the VTEP source when VXLAN gateways are required. Calculation is done using the 6-bytes of reserved ULA space, the 2-byte Fabric ID, and the node's 8-byte System ID. Derivation of the System ID varies slightly depending upon the node's location/role in the fabric and will be described in subsequent sections.

#### 6.3.1. Leaf Nodes as Gateways

Calculation is done using the 6-bytes of reserved ULA space, the 2-byte Fabric ID, and the node's 8-byte System ID.

In order for leaf nodes to derive IPv6 loopback addresses, algorithms shown in both `auto_evpn_fidsidv6loopback` (Figure 28) and `auto_evpn_v6prefixfidsid2loopback` (Figure 13) are required.

IPv4 addresses MAY be supported, but it should be noted that they have a higher likelihood of collision. The appendix contains the required `auto_evpn_fidsid2v4loopback` (Figure 27) algorithm to support IPv4 loopback derivation.

#### 6.3.2. ToF Nodes as Route Reflectors

ToF nodes acting as route reflectors MUST derive their loopback address according to the specific section describing the algorithm. Calculation is done using the 6-bytes of reserved ULA space, the 2-byte Fabric ID, and the 8-byte System ID of each elected route reflector.

In order for the ToF nodes to derive IPv6 loopbacks, the algorithms shown in both `auto_evpn_fidsidv6loopback` (Figure 28) and `auto_evpn_fidrrpref2rrloopback` (Figure 14) are required.

In order for the ToF derive the necessary prefix range to facilitate peering requests from any leaf, the algorithm shown in "`auto_evpn_fid2fabric_prefixes`" (Figure 12) is required.

A topology MUST elect at least 1 Top-of-Fabric node as an IBGP route reflector, but SHOULD elect 3. The election method varies depending upon whether the fabric is comprised of a single plane or multiple planes or if DCI gateway functionality is desired.

##### 6.3.2.1. Single Plane Route Reflector Election Procedures

Each ToF performs the election independently based on system IDs of other ToFs in the fabric obtained via southbound reflection. The route reflector election procedures are defined as follows:

1. ToF node with the highest System ID.
2. ToF node with the lowest System ID.
3. ToF node with the 2nd highest System ID.
4. etc.

This ordering is necessary to prevent a single node with either the highest or lowest System ID from triggering changes to route reflector loopback addresses as it would result in all BGP sessions dropping.

For example, if two nodes, ToF01 and ToF02 with System IDs 002c6af5a281c000 and 002c6bf5788fc000 respectively, ToF02 would be elected due to it having the highest System ID of the ToFs (002c6bf5788fc000). If a ToF determines that it is elected as route reflector, it uses the knowledge of its position in the list to derive route reflector IPv6 loopback address.

The algorithm shown in "auto\_evpn\_sids2rrs" (Figure 10) is required to accomplish this.

#### 6.3.2.1.1. DCI-GW Variations

It is beneficial for ToF-RRs requiring DCI-GW functions to be preferred over ToF-RRs that do not. As such, the "default\_acting\_auto\_evpn\_dci\_when\_tof" flag described in Appendix A.1 MUST be factored into election procedures mentioned in the previous section. Essentially, ToFs flagged as requiring DCI-GW functions, will be sorted separately from those that do not. That is to say, that ToFs requiring DCI-GW functions will always be preferred as RRs.

For example, if a fabric contains 4 ToF nodes where 2 require DCI-GW functions and the other 2 do not, the election will take place as follows:

1. ToF node (DCI) with the highest System ID.
2. ToF node (DCI) with the lowest System ID.
3. ToF node (non-DCI) with the 2nd highest System ID.
4. etc.

#### 6.3.2.2. Multiplane Route Reflector Election Procedures

As mentioned in the main RIFT [RIFT] specification, when an implementation uses multiplane fabrics, inter-ToF rings are recommended in order to facilitate northbound flooding between ToFs in different planes.

If a multiplane implementation is using Auto-EVPN, those inter-ToF rings are REQUIRED to ensure that DCI/RR election works as intended.

Each ToF performs the election independently based on system IDs of other ToFs in the other fabrics obtained from northbound flooding across the inter-ToF rings. The highest System ID from each plane will be considered the Plane ID, which is then factored into the election as follows:

1. The ToF node with the highest Plane ID, DCI bit, System ID
2. The ToF node with the lowest Plane ID, DCI bit, System ID
3. The ToF node with the 2nd highest Plane ID, DCI bit, System ID
4. etc.

This algorithm allows DCI/RRs to be split across planes for improved redundancy.

#### 6.4. Autonomous System Number

Nodes in each fabric MUST derive a private autonomous system number based on its Fabric ID so that it is unique across the fabric.

The algorithm shown in `auto_evpn_fid2private_AS` (Figure 29) is required to derive the private ASN.

#### 6.5. Router ID

Nodes MUST drive a Router ID that is based on both its System ID and Fabric ID so that it is unique to both.

The algorithm shown in `auto_evpn_sidfid2bgpid` (Figure 15) is required to derive the BGP Router ID.

#### 6.6. Cluster ID

Route reflector nodes in each fabric MUST derive a cluster ID that is based on its Fabric ID so that it is unique across the fabric.

The algorithm shown in `auto_evpn_fid2clusterid` (Figure 30) is required to derive the BGP Cluster ID.

#### 6.7. Route Target

Nodes hosting EVPN EVIs MUST derive a route target extended community based on the MAC-VRF ID for each EVI so that it is unique across the network. Route targets MUST be of type 0 as per RFC4360.

For example, if given a MAC-VRF ID of 1, the derived route target would be "target:1"

The algorithm shown in `auto_evpn_evi2rt` (Figure 16) is required to derive the Route Target community.

#### 6.8. Route Distinguisher

Nodes hosting EVPN EVIs MUST derive a type-0 route distinguisher based on its System ID and Fabric ID so that it is unique per node within a fabric.

The algorithm shown in `auto_evpn_sidfid2rd` (Figure 22) is required to derive the Route Distinguisher.

#### 6.9. EVPN MAC-VRF Services

It's obvious that applications utilizing Auto-EVPN overlay services may require a variety of layer-2 and/or layer-3 traffic considerations. Variables supporting these services are also derived based on some combination of MAC-VRF ID, Fabric ID, and other constant values. Integrated Routing and Bridging (IRB) gateway address derivation also leverages a set of constant RANDOMSEEDS (Figure 9) values that MUST be used to provide additional entropy.

In order to ensure that VLAN ID's don't collide, a single deployment SHOULD NOT exceed 6 fabrics with 7 EVIs where each EVI terminates 30 VLANs. The algorithms shown in `auto_evpn_fidevivlansvlans2desc` (Figure 20) and `auto_evpn_vlan_description_table` (Figure 19) are required to derive VLANs accordingly. An implementation MAY exceed this, but MUST indicate methods to ensure collision-free derivation and describe which VLANs are stretched across fabrics.

Lastly, Table 3 shows example derivation results for the previously mentioned scaling figures.

#### 6.9.1. Untagged Traffic in Multiple Fabrics

This section defines methods to derive unique VLAN, VNI, MAC, and gateway address values for deployments where untagged traffic is stretched across multiple fabrics.

##### 6.9.1.1. VLAN

Untagged traffic stretched across multiple fabrics MUST derive VLAN tags based on MAC-VRF ID in conjunction with a constant value.

##### 6.9.1.2. VNI

Untagged traffic stretched across multiple fabrics MUST derive VNIs based on MAC-VRF ID in conjunction with a constant value. These VNIs MUST correspond to EVPN Type-2 routes.

The algorithm shown in `auto_evpn_fidevivid2vni` (Figure 18) is required to derive VNIs for Type-2 EVPN routes.

##### 6.9.1.3. MAC Address

The MAC address MUST be a unicast address and also MUST be identical for any IRB gateways that belong to an individual bridge-domain across fabrics. The last 5-bytes MUST be a hash of the MAC-VRF ID and a constant value that is calculated using the previously mentioned random seed values.

The algorithm shown in `auto_evpn_fidevividsid2mac` (Figure 26) is required to derive MAC addresses.

##### 6.9.1.4. IPv6 IRB Gateway Address

The derived IPv6 gateway address MUST be from a ULA-scoped range that will account for the first 6-bytes. The next 5-bytes MUST be the last bytes of the derived MAC address. Finally, the remaining 7-bytes MUST be `::0001`.

The algorithm shown in `auto_evpn_fidevividsid2v6subnet` (Figure 25) is required to derive the IPv6 gateway address.

##### 6.9.1.5. IPv4 IRB Gateway Address

The derived IPv4 gateway address MUST be from a RFC1918 range, which accounts for the first octet. The next octet MUST be a hash of the MAC-VRF ID and a constant value of 1 that is calculated using the previously mentioned random seed values. Finally, the remaining 2 octets MUST be 0 and 1 respectively.

The algorithm shown in `auto_evpn_v4prefixfidevivid2v4subnet` (Figure 23) is required to derive the IPv4 gateway address. It should be noted that there is a higher likelihood of address collisions when deriving IPv4 addresses.

#### 6.9.2. Tagged Traffic in Multiple Fabrics

This section defines methods to derive unique VLAN, VNI, MAC, and gateway address values for deployments where tagged traffic is stretched across multiple fabrics.

##### 6.9.2.1. VLAN

Tagged traffic stretched across multiple fabrics MUST derive VLAN tags based on MAC-VRF ID in conjunction with a constant value.

##### 6.9.2.2. VNI

Tagged traffic stretched across multiple fabrics MUST derive VNIs based on MAC-VRF ID in conjunction with a constant value. These VNIs MUST correspond to EVPN Type-2 routes.

The algorithm shown in `auto_evpn_fidevivid2vni` (Figure 18) is required to derive VNIs for Type-2 EVPN routes.

##### 6.9.2.3. MAC Address

The MAC address MUST be a unicast address and also MUST be identical for any IRB gateways that belong to an individual bridge-domain across fabrics. The last 5-bytes MUST be a hash of the MAC-VRF ID and a constant value that is calculated using the previously mentioned random seed values.

The algorithm shown in `auto_evpn_fidevivid2mac` (Figure 26) is required to derive MAC addresses.

##### 6.9.2.4. IPv6 IRB Gateway Address

The derived IPv6 gateway address MUST be from a ULA-scoped range that will account for the first 6-bytes. The next 5-bytes MUST be the last bytes of the derived MAC address. Finally, the remaining 7-bytes MUST be `::0001`.

The algorithm shown in `auto_evpn_fidevivid2v6subnet` (Figure 25) is required to derive the IPv6 gateway address.

#### 6.9.2.5. IPv4 IRB Gateway Address

The derived IPv4 gateway address MUST be from a RFC1918 range, which accounts for the first octet. The next octet MUST be a hash of the MAC-VRF ID and a constant value of 16 that is calculated using the previously mentioned random seed values. Finally, the remaining 2 octets MUST be 0 and 1 respectively.

The algorithm shown in `auto_evpn_v4prefixfidevivid2v4subnet` (Figure 23) is required to derive the IPv4 gateway address. It should be noted that there is a higher likelihood of address collisions when deriving IPv4 addresses.

#### 6.9.3. Tagged Traffic in a Single Fabric

This section defines a method to derive unique VLAN, VNI, MAC, and gateway address values for deployments where untagged traffic is contained within a single fabric.

##### 6.9.3.1. VLAN

Tagged traffic contained to a single fabric MUST derive VLAN tags based on MAC-VRF ID and Fabric ID in conjunction with a constant value.

##### 6.9.3.2. VNI

Tagged traffic contained to a single fabric MUST derive VNIs based on MAC-VRF ID and Fabric ID in conjunction with a constant value. These VNIs MUST correspond to EVPN Type-2 routes.

The algorithm shown in `auto_evpn_fidevivid2vni` (Figure 18) is required to derive VNIs for Type-2 EVPN routes.

##### 6.9.3.3. MAC Address

The MAC address MUST be a unicast address and also MUST be identical for any IRB gateways that belong to an individual bridge-domain across fabrics. The last 5-bytes MUST be a hash of the MAC-VRF ID and a constant value that is calculated using the previously mentioned random seed values.

The algorithm shown in `auto_evpn_fidevivid2mac` (Figure 26) is required to derive MAC addresses.

#### 6.9.3.4. IPv6 IRB Gateway Address

The derived IPv6 gateway address MUST be from a ULA-scoped range, which accounts for the first 6-bytes. The next 5-bytes MUST be the last bytes of the derived MAC address. Finally, the remaining 7-bytes MUST be ::0001.

The algorithm shown in `auto_evpn_fidevividssid2v6subnet` (Figure 25) is required to derive the IPv6 gateway address.

#### 6.9.3.5. IPv4 IRB Gateway Address

The derived IPv4 gateway address MUST be from a RFC1918 range, which accounts for the first octet. The next octet MUST be a hash of the MAC-VRF ID and a constant value of 17 that is calculated using the previously mentioned random seed values. Finally, the remaining 2 octets MUST be 0 and 1 respectively.

The algorithm shown in `auto_evpn_v4prefixfidevividssid2v4subnet` (Figure 23) is required to derive the IPv4 gateway address. It should be noted that there is a higher likelihood of address collisions when deriving IPv4 addresses.

### 6.9.4. Traffic Routed to External Destinations

#### 6.9.4.1. Route Distinguisher

Nodes hosting IP Prefix routes MUST derive a type-0 route distinguisher based on its System ID and Fabric ID so that it is unique per IP-VRF and per node.

The algorithm shown in `auto_evpn_sidfid2rd` (Figure 22) is required to derive the Route Target.

#### 6.9.4.2. Route Target

Nodes hosting IP prefix routes MUST derive a route target extended community based on the MAC-VRF ID for each IP-VRF so that it is unique across the network. Route targets MUST be of type 0.

The algorithm shown in `auto_evpn_evi2rt` (Figure 16) is required to derive the Route Target community.

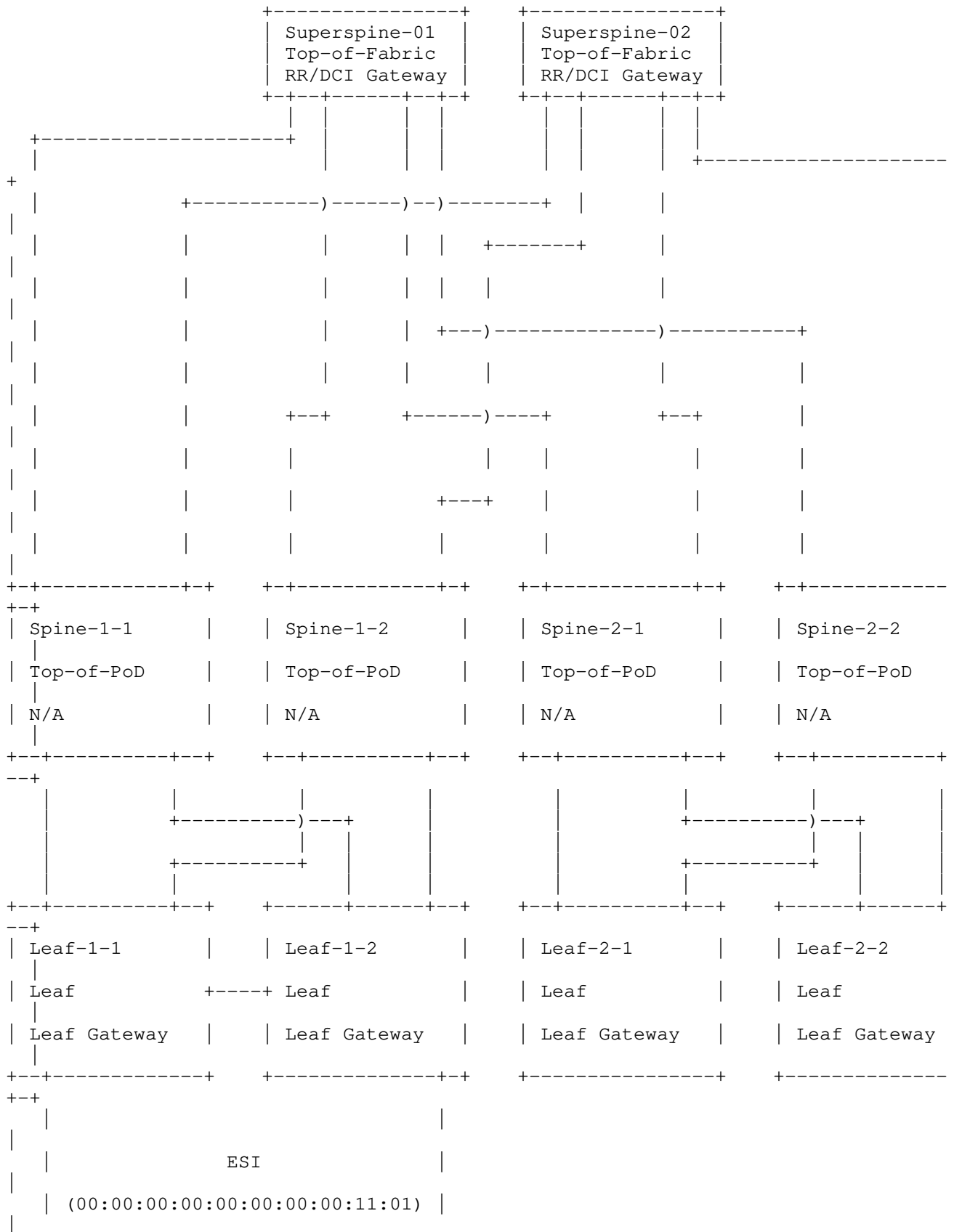


## 7. Operational Considerations

To fully realize the benefits of Auto-EVPN, it may help to describe the high-level methodology. Simply put, RIFT automatically provisions the underlay and Auto-EVPN provisions the overlay. The goal of this section is to draw clear lines between general fabric concepts, RIFT, and Auto-EVPN and how they fit into current network designs and practices.

This section also describes an set of optional Key-Value TIEs that leverages the variables that have already been derived to provide further operational enhancement to the operator.

### 7.1. RIFT Underlay and Auto-EVPN Overlay



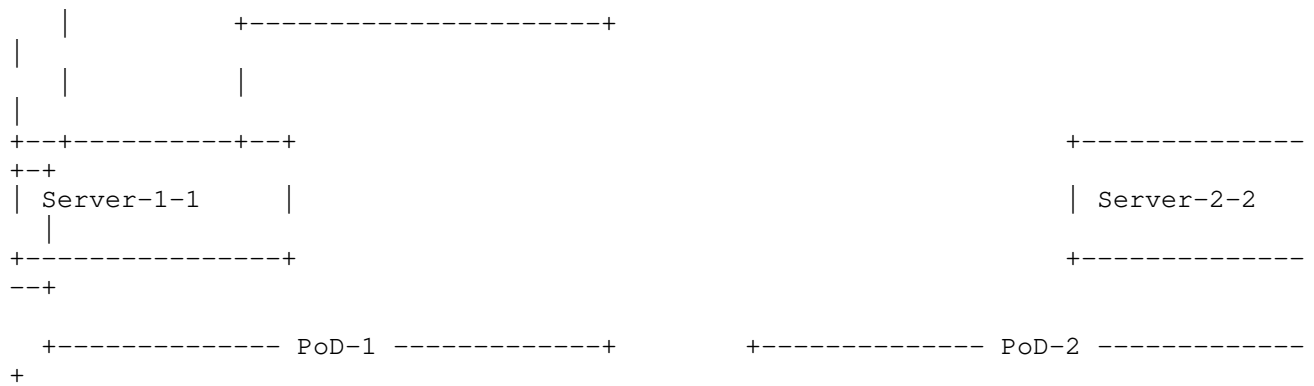


Figure 1: Auto-EVPN Example Topology

Figure 1 illustrates a typical 5-stage Clos IP fabric. Each node is labelled in such a way that conveys the following:

1. The nodes placement within the generic IP fabric.
2. The nodes role within the RIFT IP underlay.
3. The nodes role within the Auto-EVPN overlay.

Table 1 should also help further align these concepts.

Fabric Placement	RIFT Role	Auto-EVPN Role
Superspine	Top-of-Fabric	Route Reflector and/or DCI Gateway
Spine	Spine or Top- of-PoD	N/A
Leaf	Leaf	Leaf Gateway

Table 1: Role Associations

It's also important to remember that Auto-EVPN simply takes existing EVPN overlay deployment scenarios and simplifies the provisioning. Figure 2 further illustrates the resulting EVPN overlay topology.

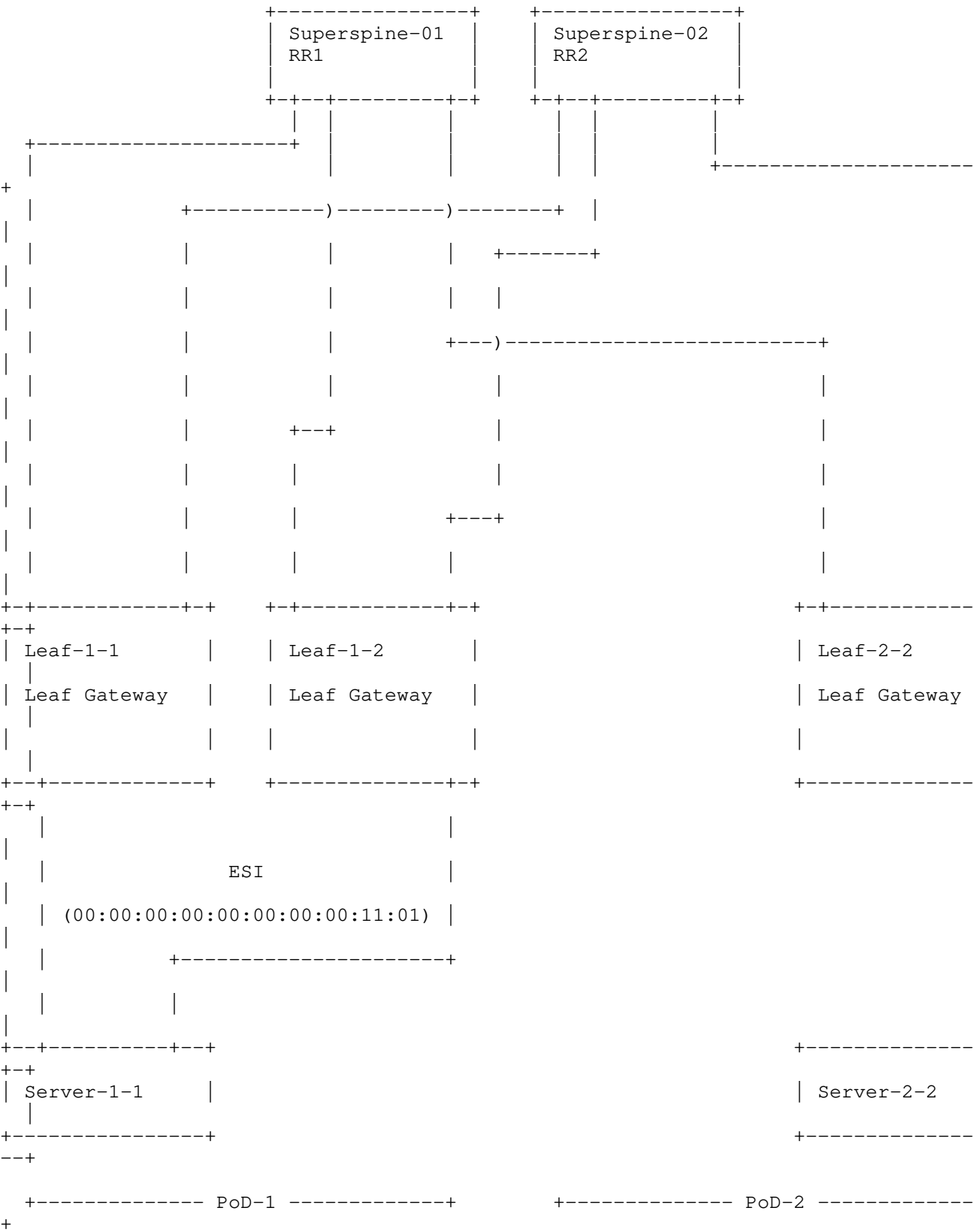


Figure 2: Auto-EVPN Overlay Topology

## 7.2. Auto-EVPN Analytics

Leaf nodes MAY optionally advertise analytics information about the Auto-EVPN fabric to ToF nodes using RIFT Key-Value TIEs. This may be advantageous in that overlay validation and troubleshooting activities can be performed on the ToF nodes.

This section requests suggested values from the RIFT Well-Known Key-Type Registry and describes their use for Auto-EVPN.

Name	Value	Description
Auto-EVPN Analytics MAC-VRF	3	Analytics describing a MAC-VRF on a particular node within a fabric.
Auto-EVPN Analytics Global	4	Analytics describing an Auto-EVPN node within a fabric.

Table 2: Requested RIFT Key Registry Values

The normative Thrift schema can be found in the appendix (Appendix A.4).

#### 7.2.1. Auto-EVPN Global Analytics Key Type

This Key Type describes node level information within the context of the Auto-EVPN fabric. The System ID of the advertising leaf node MUST be used to differentiate the node among other nodes in the fabric.

The Auto-EVPN Global Key Type MUST be advertised with the RIFT Fabric ID encoded into the 3rd and 4th bytes of the Key Identifier.

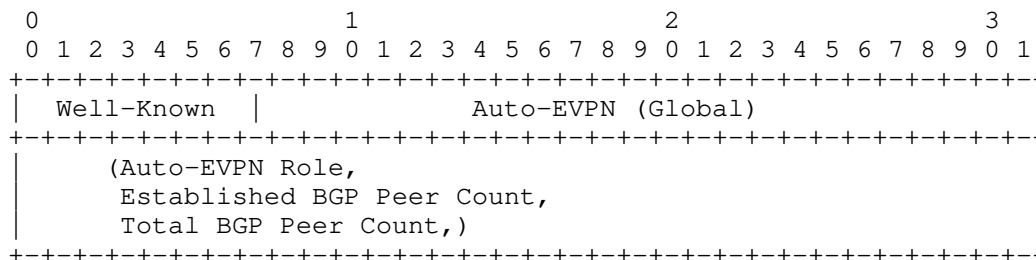


Figure 3: Auto-EVPN Global Key-Value TIE

where:

**\*Auto-EVPN Role:\***

The value indicating the node's Auto-EVPN role within the fabric.

0: Illegal value, MUST NOT be used.

1: Auto-EVPN Leaf Gateway

2: Auto-EVPN Top-of-Fabric Gateway

**\*Established BGP Session Count:\***

A 16-bit integer indicating the number of BGP sessions in the Established state.

**\*Total BGP Peer Count:\***

A 16-bit integer indicating the total number of possible BGP sessions on the local node, regardless of state.

### 7.2.2. Auto-EVPN MAC-VRF Key Type

This Key-Value structure contains information about a specific MAC-VRF within the Auto-EVPN fabric.

The Auto-EVPN MAC-VRF Key Type MUST be advertised with the Auto-EVPN MAC-VRF ID encoded into the 3rd and 4th bytes of the Key Identifier.

All values advertised in a MAC-VRF Key-Value TIE MUST represent only state of the local node.

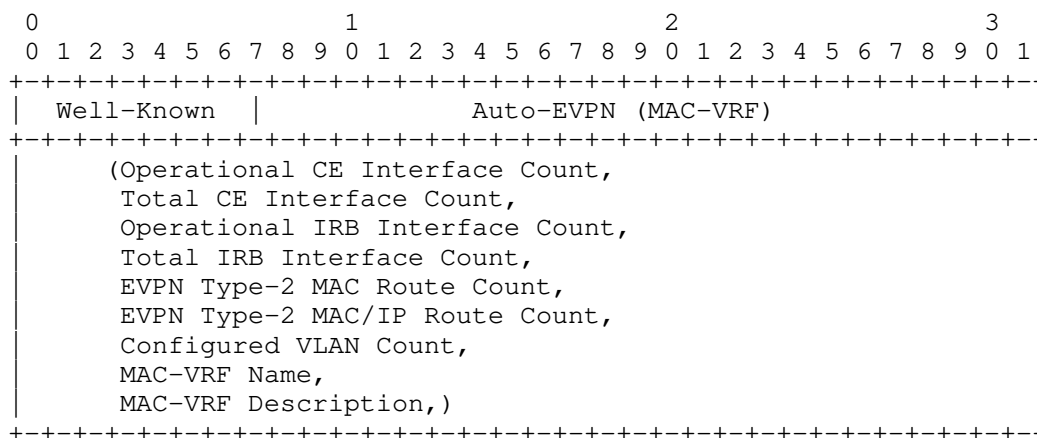


Figure 4: Auto-EVPN MAC-VRF Key-Value TIE

where:

**\*Operational Customer Edge Interface Count:\***

A 16-bit integer indicating the number of CE interfaces associated with the MAC-VRF where both administrative and operational status are "up".

**\*Total Customer Edge Interface Count:\***

A 16-bit integer indicating the total number of CE interfaces associated with the MAC-VRF regardless of interface status.



\*Operational IRB Interface Count:\*

A 16-bit integer indicating the number of IRB interfaces associated with the MAC-VRF where both administrative and operational status are "up".

\*Total IRB Interface Count:\*

A 16-bit integer indicating the total number of IRB interfaces associated with the MAC-VRF regardless of interface status.

\*EVPN Type-2 MAC Route Count:\*

A 32-bit integer indicating the total number of EVPN Type-2 MAC routes.

\*EVPN Type-2 MAC/IP Route Count:\*

A 32-bit integer indicating the total number of EVPN Type-2 MAC/IP routes.

\*VLAN Count:\*

A 16-bit integer indicating the total number configured VLANs.

\*MAC-VRF Name:\*

A string used to indicate the name of the MAC-VRF on the node.

\*MAC-VRF Description:\*

A string used to describe the MAC-VRF on the node, similar to that of an interface description.

## 8. Acknowledgements

The authors would like to thank Olivier Vandezande for some nice operational improvements for variable derivation procedures, as well as Matthew Jones and Michal Styszynski for their contributions.

## 9. Security Considerations

This document introduces no new security concerns to RIFT or other specifications referenced in this document.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC7432] Sajassi, A., Aggarwal, R., Bitar, N., Isaac, A., Uttaro, J., Drake, J., and W. Henderickx, "BGP MPLS-Based Ethernet VPN", February 2015, <<https://www.rfc-editor.org/info/rfc7432>>.
- [RIFT] Przygienda, T., Sharma, A., Thubert, P., Rijsman, B., and D. Afanasiev, "RIFT: Routing in Fat Trees", Work in Progress, draft-ietf-rift-rift-13, July 2021.
- [RIFT-KV] Head, J. and T. Przygienda, "RIFT Keys Structure and Well-Known Registry in Key Value TIE", Work in Progress, draft-head-rift-kv-registry-01, July 2021.

## Appendix A. Thrift Models

This section contains the normative Thrift models required to support Auto-EVPN. Per the main RIFT [RIFT] specification, all signed values MUST be interpreted as unsigned values.

### A.1. common.thrift

This section specifies changes to main RIFT common.thrift model.

```
...
/** EVPN Fabric ID */
typedef il6      FabricIDType

const FabricIDType undefined_fabric_id  = 0
const FabricIDType default_fabric_id    = 1

const    bool    default_acting_auto_evpn_dci_when_tof      = false

enum AutoEVPNModel {
    ERB_VLAN_BUNDLE = 0,
}

const AutoEVPNModel default_autoevpn_model = AutoEVPNModel.ERB_VLAN_BUNDLE
```

Figure 5: RIFT Common Schema for Auto-EVPN

### A.2. encoding.thrift

This section specifies changes to main RIFT encoding.thrift model.

```

struct LIEPacket {
...
    /** provides the optional ID of the configured auto-evpn fabric. */
    35: optional common.FabricIDType      fabric_id;
    /** provides optional version of EVPN ZTP as 256 * MAJOR + MINOR */
    36: optional il6                      auto_evpn_version;
...
}

struct NodeTIEElement {
...
    /** All Auto EVPN elements MUST be present in at least one node TIE in each di
    rection if auto evpn is running. */
    /** It provides optional version of EVPN ZTP as 256 * MAJOR + MINOR, if set au
    to EVPN is enabled. */
    21: optional il6                      auto_evpn_version;
    /** It provides the optional ID of the Fabric configured */
    22: optional common.FabricIDType      fabric_id = common.default_fabric
_id;
    /** provides optionally the EVPN model supported */
    25: optional common.AutoEVPNModel     auto_evpn_model = common.AutoEVPN
Model.ERB_VLAN_BUNDLE,
...
}

struct NodeCapabilities {
...
    /** provides the optional ID of the configured auto-evpn fabric. */
    10: optional bool                    auto_evpn_support = false;
...
}

struct NodeFlags {
...
    /** acting as DCI for auto-evpn, necessary for proper RR election where DCIs
    are preferred */
    10: optional bool
...
}

```

Figure 6: RIFT Encoding Schema for Auto-EVPN

## A.3. common\_evpn.thrift

This section contains the normative Auto-EVPN Thrift schema.

```

/**
    Thrift file for common AUTO EVPN definitions for RIFT

    Copyright (c) Juniper Networks, Inc., 2016-
    All rights reserved.
*/

```

```

namespace py common_evpn
namespace rs models

include "common.thrift"
include "encoding.thrift"
include "statistics.thrift"

const i8          default_evis          = 3
const i8          default_vlans_per_evi = 7

typedef i32        RouterIDType
typedef i32        ASType
typedef i32        ClusterIDType

struct EVPNAnyRole {
    1: required    common.Ipv6Address      v6_loopback,
    2: required    common.Ipv6Address      type5_v6_loopback,
    3: required    common.Ipv4Address      type5_v4_loopback,
    4: required    RouterIDType            bgp_router_id,
    5: required    ASType                  autonomous_system,
    6: required    ClusterIDType           cluster_id,
    /** prefixes to be redistributed north */
    7: required    set<common.IPPrefixType> redistribute_north,
    /** prefixes to be redistributed south */
    8: required    set<common.IPPrefixType> redistribute_south,
    /** group name for evpn auto overlay */
    9: required    string                  bgp_group_name,
    /** fabric prefixes to be advertised in rift instead of default */
    10: required   set<common.IPPrefixType> fabric_prefixes,
    /** v6 loopback prefix range, used e.g. to clean up config */
    20: required   common.Ipv6PrefixType   v6_loopback_range,
    21: required   common.Ipv6PrefixType   rr_loopback_range,
    22: required   common.Ipv6PrefixType   type5_loopback_range,
    23: required   common.Ipv4PrefixType   type5_v4_loopback_range,
    /** v6 addresses of all possible RR loopbacks in this config. Can be used for
    e.g. cleanup */
    24: required   set<common.Ipv6PrefixType> possible_elected_rrs,
}

struct PartialEVPNEVI {
    // route target per RFC4360
    1: required    CommunityType            rt_target,
    2: required    RTDistinguisherType      rt_distinguisher,
    3: required    RTDistinguisherType      rt_type5_distinguisher,
    5: required    string                    mac_vrf_name,
    6: required    VNIDType                  type5_vni,
}

struct EVPNRRRole {

```

```

        2: required    common.Ipv6Address          v6_rr_addr_loopback,
        3: required    common.Ipv6PrefixType       v6_peers_allowed_range,
        4: required    map<MACVRFNumberType, PartialEVPNEVI> evis,
    }

typedef i64          RTDistinguisherType
typedef i64          RTTargetType
typedef i16          MACVRFNumberType

typedef i16          VLANIDType
typedef binary       MACType

typedef i16          UnitType

struct IRBType {
    1: required    string          name,
    2: required    UnitType        unit,
    /// constant
    3: required    MACType          mac,
    /// contains address of the gateway as well
    4: optional    common.Ipv6PrefixType    v6_subnet,
    /// contains address of the gateway as well
    5: optional    common.Ipv4PrefixType    v4_prefix,
}

typedef i32          VNIType

struct VLANType {
    1: optional    VLANIDType      id,
    2: required    string          name,
    3: optional    IRBType         irb,
    5: optional    bool            stretched = false,
    6: optional    bool            is_native = false,
}

struct CEInterfaceType {
    2: optional    common.IEEE802_1ASTimestampType    moved_to_ce,
    // we may not be able to obtain it in case of internal errors
    3: optional    string          platform_interface_name,
}

typedef i64          CommunityType

struct EVPNEVI {
    // route target per RFC4360
    1: required    CommunityType    rt_target,
    2: required    RTDistinguisherType    rt_distinguisher,
    3: required    RTDistinguisherType    rt_type5_distinguisher,
}

```

```

        4: required    string                                mac_vrf_name,
        // fabric unique 24 bits VNI on non-stretch, otherwise unique across fabrics
        5: required    map<VNIType, VLANType>                vlans,
        6: required    VNIType                                type5_vni,
    }

    struct EVPNLeafRole {
        1: required    set<common.IPv6Address>                rrs,
        2: required    map<MACVRFNumberType, EVPNEVI>         evis,
        3: optional    map<common.LinkIDType,
                        CEInterfaceType>                        ce_interfaces,

        5: optional    binary                                  leaf_unique_lacp_system_i
d,
        6: optional    binary                                  fabric_unique_lacp_system
_id,
    }

    /// structure to indicate EVPN roles assumed and their variables for
    /// external platform to configure itself accordingly. Presence of
    /// according structure indicates that the role is assumed.
    struct EVPNRoles {
        1: required    EVPNAnyRole                            generic,
        2: optional    EVPNRRRole                             route_reflector,
        3: optional    EVPNLeafRole                           leaf,
    }

    const common.TimeIntervalInSecType    default_leaf_delay = 120
    const common.TimeIntervalInSecType    default_interface_ce_delay = 180
    /// default delay before AUTOEVPN FSM starts to compute anything
    const common.TimeIntervalInSecType    default_AUTOEVPN_startup_delay = 60

```

Figure 7: Auto-EVPN Common Thrift Schema

#### A.4. auto\_evpn\_kv.thrift

This section contains the normative Auto-EVPN Analytics Thrift schema.

```

include "common.thrift"

namespace py auto_evpn_kv
namespace rs models

/** We don't need the full role structure, only an indication of the node's basic
    role */
enum AutoEVPNRole {
    ILLEGAL                = 0,
    auto_evpn_leaf_erb     = 1,
    auto_evpn_tof_gw       = 2,
}

```

```

enum    KVTypes {
    OUI      = 1,
    WellKnown = 2,
}

const i8      AutoEVPNWellKnownKeyType = 1
typedef i32    AutoEVPNKeyIdentifier
typedef i16    AutoEVPNCOUNTERType
typedef i32    AutoEVPNLongCounterType

const i8      GlobalAutoEVPNTelemetryKV = 4
const i8      AutoEVPNTelemetryKV      = 3

/** Per the according RIFT draft the key comes from the well known space.
    Part of the key is used as Fabric-ID.

    1st      byte  MUST be = "Well-Known"
    2nd      byte  MUST be = "Global Auto-EVPN Telemetry KV",
    3rd/4th bytes MUST be = FabricIDType
*/
struct AutoEVPNTelemetryGlobalKV {
    /** Only values that the ToF cannot derive itself should be flooded. */
    1: required  set<AutoEVPNRole>          auto_evpn_roles,

    /** Established BGP peer count (for Auto-EVPN)
    2: optional  AutoEVPNCOUNTERType        established_bgp_peer_count,

    /** Total BGP peer count (for Auto-EVPN)
    3: optional  AutoEVPNCOUNTERType        total_bgp_peer_count,
}

/** Per the according RIFT draft the key comes from the well known space.
    Part of the key is used as MAC-VRF number.

    1st      byte  MUST be = "Well-Known"
    2nd      byte  MUST be = indicates "Auto-EVPN Telemetry KV",
    3rd/4th bytes MUST be = MACVRFNumberType
*/
struct AutoEVPNTelemetryMACVRFKV {
    /** Active CE interface count (up/up)
    1: optional  AutoEVPNCOUNTERType        active_ce_interfaces,

    /** Total CE interface count
    2: optional  AutoEVPNCOUNTERType        total_ce_interfaces,

    /** Active IRB interface count (up/up)
    3: optional  AutoEVPNCOUNTERType        active_irb_interfaces,

```

```

/** Total IRB interface count
4: optional  AutoEVPNCounterType          total_irb_interfaces,

/** Local EVPN Type-2 MAC route count
5: optional  AutoEVPNLongCounterType      local_evpn_type2_mac_routes,

/** Local EVPN Type-2 MAC/IP route count
6: optional  AutoEVPNLongCounterType      local_evpn_type2_mac_ip_routes,

/** number of configured VLANs */
7: optional  il6                          configured_vlans,

/** optional human readable name */
8: optional  string                       name,

/** optional human readable string describing the MAC-VRF */
9: optional  string                       description,
}

```

Figure 8: Auto-EVPN Key-Value Thrift Schema

## Appendix B. Auto-EVPN Variable Derivation

### B.1. Variable Derivation Functions

This section contains the normative derivation procedures required to support Auto-EVPN.

```

/// indicates how many RRs we're computing in AUTO EVPN
pub const MAX_AUTO_EVPN_RRS: usize = 3;
/// indicates the fabric has no ID, used in computations to omit effects of fabric ID
pub const NO_FABRIC_ID: FabricIDType = 0;
/// invalid MACVRF number, MACVRFs start from 1
pub const NO_MACVRF: MACVRFNumberType = 0;
/// first MACVRF
pub const MIN_MACVRF : MACVRFNumberType = 1;

/// unique v6 prefix for all nodes starts with this
pub fn auto_evpn_v6pref(fid: FabricIDType) -> String {
    format!("FD00::{:04X}:A1", fid)
}
/// how many bytes in a v6pref for different purposes
pub const AUTO_EVPN_V6PREFLEN: usize = 8 * 5;
/// unique v6 prefix for route reflector purposes starts like this
pub fn auto_evpn_v6rrpref(fid: FabricIDType) -> String {
    format!("FD00::{:04X}:A2", fid)
}
/// unique v6 prefix for type-5 purposes starts like this

```



```
pub fn auto_evpn_v6t5pref(fid: FabricIDType) -> String {
    format!("FD00:{:04X}:A3", fid)
}
/// unique v6 prefix for IRB prefix purposes
pub fn auto_evpn_v6irbpref(fid: FabricIDType) -> String {
    format!("FD00:{:04X}:A4", fid)
}
/// 2 bytes of prefix, then fabric ID, then another byte
pub const AUTO_EVPN_V6_FABPREFIXLEN: usize = 16 + 16 + 8;
/// unique v4 prefix for IRB purposes
pub const AUTO_EVPN_V4IRBPREF: &str = "10";

/// per RFC magic
const RT_TARGET_HIGH: CommunityType = 0;
const RT_TARGET_LOW: CommunityType = 0;

/// first available VLAN number
pub const FIRST_VLAN: UnsignedVLANIDType = 1;
/// maximum vlan number one less than maximum to use as bitmask
pub const MAX_VLAN: UnsignedVLANIDType = 4095;
/// constant VLAN shift
pub const FIRST_VLAN_SHIFT: UnsignedVLANIDType = NATIVE_VLAN + 1;
/// NATIVE VLAN number
pub const NATIVE_VLAN: UnsignedVLANIDType = 1;

/// abstract description of VLAN properties for a derived VLAN
pub struct VLANDescription {
    pub vlan_id: UnsignedVLANIDType,
    pub name: String,
    /// can this VLAN be stretched across multiple fabrics
    pub stretchable: bool,
    pub native: bool,
}

/// maximum number of VLANs per MACVRF
pub const MAX_VLANS_PER_EVI: usize = 30;

/// maximum number of EVIs
pub const MAX_EVIS: MACVRFNumberType = 7;

pub type VLANStretchableType = bool;
pub type VLANNativeType = bool;

pub type UnsignedVNIDType = u32;
pub type UnsignedFabricIDType = u16;

pub type UnsignedUnitType = u16;
pub type UnsignedVLANIDType = u16;
```

```

pub type UnsignedRTDistinguisherType = u64;

pub const EXTRATYPE5_RD_DISTINGUISHER: u32 = 0xffff_ffff;

/// high bits of type 5 VNI
const TYPE5VNIHIGH: UnsignedVNIType = 0x0080_0000;
/// bitmask for type 2 VNI
const TYPE2VNIMASK: UnsignedVNIType = 0x00ff_ffff ^ TYPE5VNIHIGH;

/// random seeds used in several algorithms to increase entropy
pub const RANDOMSEEDS: [u64; 4] = [
    27008318799u64,
    67438371571,
    37087353685,
    88675895388,
];

```

Figure 9: auto\_evpn\_const\_structs\_type

```

/// function sorts vector of (is_dci, systemID) first,
/// splits of the DCIs from the non-DCIs and sorts them
/// followed by a shuffle taking largest/smallest/2nd largest/2nd smallest.
/// Ultimately both are merged which prefers the DCIs while
/// still making sure that the election is stable with a system ID joining
/// as smallest/largest.
pub(crate) fn auto_evpn_sids2rrs(v: Vec<(bool, UnsignedSystemID)>)
    -> Vec<UnsignedSystemID> {
    let (dcis, nondcis): (Vec<(bool, UnsignedSystemID)>, Vec<(bool, UnsignedSystemID)>) =
        v.into_iter().partition(|(dci, _)| *dci);

    vec![dcis, nondcis]
        .into_iter()
        .flat_map(|mut v| {
            v.par_sort();
            if v.len() > 2 {
                let mut s = v.split_off(v.len() / 2);
                s.reverse();
                interleave(v.into_iter(), s.into_iter())
                    .collect::<Vec<_>>()
                    .into_iter()
            } else {
                v.into_iter()
            }
        })
        .map(|(_, sid)| sid)
        .collect()
}

```

Figure 10: auto\_evpn\_sids2rrs

```
pub(crate) fn auto_evpn_v62octets(a: Ipv6Addr) -> Vec<u8> {
    a.octets().iter().cloned().collect()
}
```

Figure 11: auto\_evpn\_v62octets

```
/// fabric prefixes derived instead of advertising default on the fabric to allow
/// for default route on ToF or leaves
pub fn auto_evpn_fid2fabric_prefixes(fid: FabricIDType) -> Result<Vec<IPPrefixType>
e>, ServiceErrorType> {
    vec![
        (auto_evpn_fidsidv6loopback(fid, ILLEGAL_SYSTEM_I_D as _), AUTO_EVPN_V6PR
EFLEN),
        (auto_evpn_fidrrpref2rrloopback(fid, ILLEGAL_SYSTEM_I_D as _), AUTO_EVPN_
V6PREFLEN),
    ]
    .into_iter()
    .map(|(p, _)|
        match p {
            Ok(_) => Ok(
                IPPrefixType::Ipv6prefix(
                    Ipv6PrefixType {
                        address: auto_evpn_v62octets(p?),
                        prefixlen: AUTO_EVPN_V6PREFLEN as _,
                    },
                ),
            Err(e) => Err(e),
        }
    )
    .collect::<Result<Vec<_>, _>>>()
}
```

Figure 12: auto\_evpn\_fid2fabric\_prefixes

```

/// local address with encoded fabric ID and system ID for collision free identifiers. Basis
/// for several different prefixes.
pub fn auto_evpn_v6prefixfidsid2loopback(v6pref: &str, fid: FabricIDType,
                                         sid: UnsignedSystemID) -> Result<Ipv6Addr,
                                         ServiceErrorType> {
    assert!(fid != UNDEFINED_FABRIC_ID);
    let a = format!("{}", 00::{}",
                    v6pref,
                    sid.to_ne_bytes()
                        .iter()
                        .chunks(2)
                        .into_iter()
                        .map(|chunk|
                            chunk.fold(0u16, |v, n| (v << 8) | *n as u16))
                        .map(|v| format!("{:04X}", v))
                        .collect::<Vec<_>>()
                        .into_iter()
                        .join(":")
                    );

    Ipv6Addr::from_str(&a)
        .map_err(|_| ServiceErrorType::INTERNALRIFTERROR)
}

```

Figure 13: auto\_evpn\_v6prefixfidsid2loopback

```

/// auto evpn V6 loopback for RRs
pub fn auto_evpn_fidrrpref2rrloopback(fid: FabricIDType,
                                       preference: u8) -> Result<Ipv6Addr, ServiceErrorType> {
    auto_evpn_v6prefixfidsid2loopback(&auto_evpn_v6rrpref(fid), fid, (1 + preference) as _)
}

```

Figure 14: auto\_evpn\_fidrrpref2rrloopback

```

/// auto evpn BGP router ID
pub fn auto_evpn_sidfid2bgpid(fid: FabricIDType, sid: UnsignedSystemID) -> u32 {
    assert!(fid != 0);
    let hs: u32 = ((sid & 0xffff_ffff_0000_0000) >> 32) as _;
    let mut ls: u32 = (sid & 0xffff_ffff) as _;
    ls = ls.rotate_right(7) ^ (fid as u32).rotate_right(13);
    max(1, hs ^ ls) // never a 0
}

```

Figure 15: auto\_evpn\_sidfid2bgpid

```

/// route target bytes are type0/0 and then add EVI
pub fn auto_evpn_evi2rt(evi: MACVRFNumberType) -> CommunityType {
    let wideevi = (evi + 1) as CommunityType;

    (RT_TARGET_HIGH << (64 - 8)) | (RT_TARGET_LOW << 64 - 16) |
    ((wideevi) << 17) |
    ((wideevi))
}

```

Figure 16: auto\_evpn\_evi2rt

```

/// type-5 VNI for an EVI
pub fn auto_evpn_fidevi2type5vni(fid: FabricIDType, evi: MACVRFNumberType) -> UnsignedVNIType {
    TYPE5VNIHIGH | auto_evpn_fidevivid2vni(fid, evi, 0, false)
}

```

Figure 17: auto\_evpn\_fidevi2type5vni

```

/// type-2 VNI for a specific VLAN
pub fn auto_evpn_fidevivid2vni(fid: FabricIDType, evi: MACVRFNumberType, vlanid: VLANIDType, stretchable: bool) -> UnsignedVNIType {
    let rfid = if stretchable {
        NO_FABRIC_ID as _
    } else {
        fid as UnsignedVNIType
    };

    let revi = evi as UnsignedVNIType;
    let rvlan = vlanid as UnsignedVNIType;
    // mask out high bits, VNI is only 24 bits
    TYPE2VNIMASK &
    (
        rfid.rotate_left(16) ^
        revi.rotate_left(12) ^
        rvlan
    )
}

```

Figure 18: auto\_evpn\_fidevivid2vni

```

/// maximum VLANs per EVI supported by auto evpn when deriving
pub fn auto_evpn_vlan_description_table<'a>(vlans: usize)
    -> Result<&'a [(UnsignedVLANIDType, V
LANStretchableType, VLANNativeType)], ServiceErrorType> {
    // up to 15 vlans can be activated
    const VLANSARRAY: [(UnsignedVLANIDType, bool, bool); MAX_VLANS_PER_EVI] = [
        (NATIVE_VLAN, true, true, ),
        (FIRST_VLAN_SHIFT, true, false, ),
        (FIRST_VLAN_SHIFT + 1, true, false, ),
        (FIRST_VLAN_SHIFT + 2, true, false, ),
        (FIRST_VLAN_SHIFT + 3, true, false, ),
        (FIRST_VLAN_SHIFT + 4, true, false, ),
        (FIRST_VLAN_SHIFT + 5, true, false, ),
        (FIRST_VLAN_SHIFT + 6, true, false, ),
        (FIRST_VLAN_SHIFT + 7, true, false, ),
        (FIRST_VLAN_SHIFT + 8, false, false, ),
        (FIRST_VLAN_SHIFT + 9, false, false, ),
        (FIRST_VLAN_SHIFT +10, false, false, ),
        (FIRST_VLAN_SHIFT +11, false, false, ),
        (FIRST_VLAN_SHIFT +12, false, false, ),
        (FIRST_VLAN_SHIFT +13, false, false, ),
        (FIRST_VLAN_SHIFT +14, false, false, ),
        (FIRST_VLAN_SHIFT +15, false, false, ),
        (FIRST_VLAN_SHIFT +16, false, false, ),
        (FIRST_VLAN_SHIFT +17, false, false, ),
        (FIRST_VLAN_SHIFT +18, false, false, ),
        (FIRST_VLAN_SHIFT +19, false, false, ),
        (FIRST_VLAN_SHIFT +20, false, false, ),
        (FIRST_VLAN_SHIFT +21, false, false, ),
        (FIRST_VLAN_SHIFT +22, false, false, ),
        (FIRST_VLAN_SHIFT +23, false, false, ),
        (FIRST_VLAN_SHIFT +24, false, false, ),
        (FIRST_VLAN_SHIFT +25, false, false, ),
        (FIRST_VLAN_SHIFT +26, false, false, ),
        (FIRST_VLAN_SHIFT +27, false, false, ),
        (FIRST_VLAN_SHIFT +28, false, false, ),
    ];

    if vlans > VLANSARRAY.len() {
        return Err(ServiceErrorType::INVALIDPARAMETERVALUE)
    }

    Ok(&VLANSARRAY[..vlans])
}

```

Figure 19: auto\_evpn\_vlan\_description\_table

```

const fn num_bits<T>() -> usize { std::mem::size_of::<T>() * 8 }

fn log2(x: u32) -> u32 {
    assert!(x > 0);
    num_bits::<u32>() as u32 - x.leading_zeros() - 1
}

/// delivers the vlan description that can be used to generate vlans for a
/// specific fabric ID and a MACVRF number
pub fn auto_evpn_fidevivlansvlans2desc(fid: UnsignedFabricIDType, macvrf: MACVRFN
umberType,
                                     vlans: usize) -> Vec<VLANDescription> {
    assert!(NO_MACVRF != macvrf);

    // abstract description of derived VLANs
    let vlan_table = auto_evpn_vlan_description_table(vlans)
        .expect("vlan table in AUTO EVPN incorrect");

    let vlanshift = log2(vlan_table
        .iter()
        .map(|(vl, _, _)| *vl as usize)
        .max()
        .expect("vlan table in AUTO EVPN incorrect")
        .checked_next_power_of_two()
        .expect("vlan table in AUTO EVPN incorrect")
        as u32);

    vlan_table
        .iter()
        .map(move |(vid, stretch, native_)| {
            let stretchedfid = if !stretch {
                fid
            } else {
                NO_FABRIC_ID as _
            };

            let reducedmacvrf = macvrf - MIN_MACVRF;

            // we shift fid & evi same amount to extinguish them possibly
            let fidandevishift = vlanshift + 1;
            let mut vlan_id = *vid ^ stretchedfid
                .rotate_left(fidandevishift) as UnsignedVLANIDType;
            // leave space for VLANs in the encoding
            vlan_id ^= reducedmacvrf.rotate_left(fidandevishift) as UnsignedVLANI
DType;

            vlan_id %= MAX_VLAN;
            vlan_id = max(1, vlan_id);

            VLANDescription {

```

```

        vlan_id: vlan_id as _,
        name: format!("V{}", vlan_id),
        stretchable: *stretch,
        native: *native_,
    }
})
.collect()
}

```

Figure 20: auto\_evpn\_fidevivlansvlans2desc

```

/// IRB interface number.
/// fid/evi combination shifted up to not interfere with the VLAN-ID
/// and then add the VLAN-ID
pub fn auto_evpn_fidevid2irb(_fid: FabricIDType, _evi: MACVRFNumberType, vid: V
LANIDType) -> UnsignedUnitType {

    assert!(NO_MACVRF != _evi);

    // VLAN collision function is collision free to the point we can just ignore
EVI
    // and assign IRB interface number to be same as VLAN which simplifies deploy
ment
    let mut v: UnsignedUnitType = 0;

    v = v.wrapping_add(vid as UnsignedVLANIDType);
    max(1, v % (UnsignedUnitType::MAX - 1))
}

```

Figure 21: auto\_evpn\_fidevid2irb

```

/// route distinguisher derivation
pub fn auto_evpn_sidfid2rd(sid: UnsignedSystemID, fid: UnsignedFabricIDType, extr
a: u32) -> UnsignedRTDistinguisherType {
    // generate type 0 route distinguisher, first 2 bytes 0 and then 6 bytes
    assert!(fid != NO_FABRIC_ID as _);
    // shift the 2 bytes we loose
    let convsid = sid as UnsignedRTDistinguisherType;
    let hs = ((sid & 0xffff_0000_0000_0000) >> 32) as UnsignedRTDistinguisherType
;
    let mut ls: UnsignedRTDistinguisherType = convsid & 0x0000_ffff_ffff_ffff;
    ls ^= hs;
    ls ^= (fid as UnsignedRTDistinguisherType).rotate_left(16);
    ls ^= extra as UnsignedRTDistinguisherType;
    ls
}

```

Figure 22: auto\_evpn\_sidfid2rd



```

/// v4 subnet derivation
pub fn auto_evpn_v4prefixfidevividsid2v4subnet(v4pref: &str, fid: FabricIDType,
                                              evi: MACVRFNumberType, vid: VLANID
Type,
                                              sid: UnsignedSystemID) -> Result<I
Pv4PrefixType, ServiceErrorType> {

    assert!(NO_MACVRF != evi);

    // fid can be 0 for stretched v4subnets
    let mut sub = evi.to_ne_bytes().iter()
        .fold((RANDOMSEEDS[0] & 0xff) as u8, |r, e| r.rotate_left(1) ^ e.rotate_r
ight(1));
    sub ^= fid.to_ne_bytes().iter()
        .fold((RANDOMSEEDS[1] & 0xff) as u8, |r, e| r.rotate_left(2) ^ e.rotate_r
ight(1));
    sub ^= vid.to_ne_bytes().iter()
        .fold((RANDOMSEEDS[2] & 0xff) as u8, |r, e| r.rotate_left(3) ^ e.rotate_r
ight(1));

    let subnet = sub % 254; // make sure we don't show multicast subnet

    let _host = sid.to_ne_bytes().iter()
        .fold(0u16, |r, e| r.rotate_left(3) ^ e.rotate_right(3) as u16);

    let a = format!("{}",
                    v4pref,
                    subnet,
                    0,
                    1,
    );

    Ok(
        IPv4PrefixType {
            address: Ipv4Addr::from_str(&a)
                .map_err(|_| {
                    ServiceErrorType::INTERNALRIFTEERROR
                })?
                .octets()
                .iter()
                .fold(0u32, |v, nv| v << 8 | (*nv as u32)) as Ipv4Address
            ,
            prefixlen: 16,
        }
    )
}

```

Figure 23: auto\_evpn\_v4prefixfidevividsid2v4subnet

```

/// generic v6 bytes derivation used for different purposes
pub fn auto_evpn_v6hash(fid: FabricIDType, evi: MACVRFNumberType, vid: VLANIDType
, sid: UnsignedSystemID)
    -> [u8; 8] {

    let mut sub = evi.to_ne_bytes().iter()
        .fold(RANDOMSEEDS[3], |r, e| r.rotate_left(6) ^ e.rotate_right(4) as u64)
;
    sub ^= fid.to_ne_bytes().iter()
        .fold(RANDOMSEEDS[0], |r, e| r.rotate_left(6) ^ e.rotate_right(4) as u64)
;
    sub ^= vid as u64;
    sub ^= sid;

    sub.to_ne_bytes()
}

```

Figure 24: auto\_evpn\_v6hash

```

/// v6 subnet derivation
pub fn auto_evpn_fidevividsid2v6subnet(fid: FabricIDType, evi: MACVRFNumberType,
    vid: VLANIDType,
    sid: UnsignedSystemID) -> Result<IPv6Prefix
xType, ServiceErrorType> {

    assert!(NO_MACVRF != evi);

    let sb = auto_evpn_v6hash(fid, evi, vid, sid);

    let a = format!("{:02X}{:02X}{:02X}{:02X}{:02X}{:02X}::1",
        auto_evpn_v6irbpref(fid),
        sb[3] ^ sb[0],
        sb[4] ^ sb[1],
        sb[6],
        sb[7],
        sb[5],
        sb[2],
    );

    Ok(IPv6PrefixType {
        address: Ipv6Addr::from_str(
            &a)
        .map_err(|_| {
            ServiceErrorType::INTERNALRIFTERROR
        })?
        .octets()
        .to_vec(),
        prefixlen: 64,
    })
}

```

Figure 25: auto\_evpn\_fidevividsid2v6subnet

```

/// MAC address derivation for IRB
pub fn auto_evpn_fidevividssid2mac(fid: FabricIDType, evi: MACVRFNumberType,
                                   vid: VLANIDType, sid: UnsignedSystemID) -> Vec<
u8> {

    let sb = auto_evpn_v6hash(fid, evi, vid, sid);

    vec![0x02,
        sb[3] ^ sb[0],
        sb[4] ^ sb[1],
        sb[6],
        sb[7],
        sb[5] ^ sb[2],
    ]
}

```

Figure 26: auto\_evpn\_fidevividssid2mac

```

/// v4 loopback address derivation for every node in auto-evpn, returns address a
nd
/// subnet mask length
pub fn auto_evpn_fidsid2v4loopback(fid: FabricIDType, sid: UnsignedSystemID) -> (
IPv4Address, u8) {
    let mut derived = sid.to_ne_bytes().iter()
        .fold(0 as IPv4Address, |p, e| (p << 4) ^ (*e as IPv4Address));
    derived ^= fid as IPv4Address;
    // use the byte we loose for entropy
    derived ^= derived >> 24;
    // and sanitize for loopback range, we nuke 9 bits out
    derived &= 0x007f_ffff;

    let m = ((127 as IPv4Address) << 24) | derived;
    (m as _, 9)
}

```

Figure 27: auto\_evpn\_fidsid2v4loopback

```

/// V6 loopback derivation for every node in auto-evpn
pub fn auto_evpn_fidsidv6loopback(fid: FabricIDType,
                                   sid: UnsignedSystemID) -> Result<Ipv6Addr, Serv
iceErrorType> {
    auto_evpn_v6prefixfidsid2loopback(&auto_evpn_v6pref(fid), fid, sid)
}

```

Figure 28: auto\_evpn\_fidsidv6loopback

```
#[allow(non_snake_case)]
pub fn auto_evpn_fid2private_AS(fid: FabricIDType) -> u32 {
    assert!(fid != NO_FABRIC_ID);
    // range 42000000000-4294967294
    const DIFF: u32 = 4_294_967_294 - 4_200_000_000;
    64496 + ((fid as u32) << 3) % DIFF
}
```

Figure 29: auto\_evpn\_fid2private\_AS

```
pub fn auto_evpn_fid2clusterid(fid: FabricIDType) -> u32 {
    auto_evpn_fid2private_AS(fid)
}
```

Figure 30: auto\_evpn\_fid2clusterid

## B.2. Variable Derivation Results

This section contains functional variable derivation results that can be used as a confirmation that an implementation conforms to procedures in this document.

Fabric ID	MAC-VRF ID	VLAN ID	Stretched	VNI	IRB
1	1	1	Y	4097	1
1	1	2	Y	4098	2
1	1	3	Y	4099	3
1	1	4	Y	4100	4
1	1	5	Y	4101	5
1	1	6	Y	4102	6
1	1	7	Y	4103	7
1	1	8	Y	4104	8
1	1	9	Y	4105	9
1	1	74	N	69706	74
1	1	75	N	69707	75
1	1	76	N	69708	76

1	1	77	N	69709	77
1	1	78	N	69710	78
1	1	79	N	69711	79
1	1	80	N	69712	80
1	1	81	N	69713	81
1	1	82	N	69714	82
1	1	83	N	69715	83
1	1	84	N	69716	84
1	1	85	N	69717	85
1	1	86	N	69718	86
1	1	87	N	69719	87
1	1	88	N	69720	88
1	1	89	N	69721	89
1	1	90	N	69722	90
1	1	91	N	69723	91
1	1	92	N	69724	92
1	1	93	N	69725	93
1	1	94	N	69726	94
1	2	65	Y	8257	65
1	2	66	Y	8258	66
1	2	67	Y	8259	67
1	2	68	Y	8260	68
1	2	69	Y	8261	69
1	2	70	Y	8262	70

1	2	71	Y	8263	71
1	2	72	Y	8264	72
1	2	73	Y	8265	73
1	2	10	N	73738	10
1	2	11	N	73739	11
1	2	12	N	73740	12
1	2	13	N	73741	13
1	2	14	N	73742	14
1	2	15	N	73743	15
1	2	16	N	73744	16
1	2	17	N	73745	17
1	2	18	N	73746	18
1	2	19	N	73747	19
1	2	20	N	73748	20
1	2	21	N	73749	21
1	2	22	N	73750	22
1	2	23	N	73751	23
1	2	24	N	73752	24
1	2	25	N	73753	25
1	2	26	N	73754	26
1	2	27	N	73755	27
1	2	28	N	73756	28
1	2	29	N	73757	29
1	2	30	N	73758	30

1	3	129	Y	12417	129
1	3	130	Y	12418	130
1	3	131	Y	12419	131
1	3	132	Y	12420	132
1	3	133	Y	12421	133
1	3	134	Y	12422	134
1	3	135	Y	12423	135
1	3	136	Y	12424	136
1	3	137	Y	12425	137
1	3	202	N	78026	202
1	3	203	N	78027	203
1	3	204	N	78028	204
1	3	205	N	78029	205
1	3	206	N	78030	206
1	3	207	N	78031	207
1	3	208	N	78032	208
1	3	209	N	78033	209
1	3	210	N	78034	210
1	3	211	N	78035	211
1	3	212	N	78036	212
1	3	213	N	78037	213
1	3	214	N	78038	214
1	3	215	N	78039	215
1	3	216	N	78040	216

1	3	217	N	78041	217
1	3	218	N	78042	218
1	3	219	N	78043	219
1	3	220	N	78044	220
1	3	221	N	78045	221
1	3	222	N	78046	222
1	4	193	Y	16577	193
1	4	194	Y	16578	194
1	4	195	Y	16579	195
1	4	196	Y	16580	196
1	4	197	Y	16581	197
1	4	198	Y	16582	198
1	4	199	Y	16583	199
1	4	200	Y	16584	200
1	4	201	Y	16585	201
1	4	138	N	82058	138
1	4	139	N	82059	139
1	4	140	N	82060	140
1	4	141	N	82061	141
1	4	142	N	82062	142
1	4	143	N	82063	143
1	4	144	N	82064	144
1	4	145	N	82065	145
1	4	146	N	82066	146



1	4	147	N	82067	147
1	4	148	N	82068	148
1	4	149	N	82069	149
1	4	150	N	82070	150
1	4	151	N	82071	151
1	4	152	N	82072	152
1	4	153	N	82073	153
1	4	154	N	82074	154
1	4	155	N	82075	155
1	4	156	N	82076	156
1	4	157	N	82077	157
1	4	158	N	82078	158
1	5	257	Y	20737	257
1	5	258	Y	20738	258
1	5	259	Y	20739	259
1	5	260	Y	20740	260
1	5	261	Y	20741	261
1	5	262	Y	20742	262
1	5	263	Y	20743	263
1	5	264	Y	20744	264
1	5	265	Y	20745	265
1	5	330	N	86346	330
1	5	331	N	86347	331
1	5	332	N	86348	332

1	5	333	N	86349	333
1	5	334	N	86350	334
1	5	335	N	86351	335
1	5	336	N	86352	336
1	5	337	N	86353	337
1	5	338	N	86354	338
1	5	339	N	86355	339
1	5	340	N	86356	340
1	5	341	N	86357	341
1	5	342	N	86358	342
1	5	343	N	86359	343
1	5	344	N	86360	344
1	5	345	N	86361	345
1	5	346	N	86362	346
1	5	347	N	86363	347
1	5	348	N	86364	348
1	5	349	N	86365	349
1	5	350	N	86366	350
1	6	321	Y	24897	321
1	6	322	Y	24898	322
1	6	323	Y	24899	323
1	6	324	Y	24900	324
1	6	325	Y	24901	325
1	6	326	Y	24902	326

1	6	327	Y	24903	327
1	6	328	Y	24904	328
1	6	329	Y	24905	329
1	6	266	N	90378	266
1	6	267	N	90379	267
1	6	268	N	90380	268
1	6	269	N	90381	269
1	6	270	N	90382	270
1	6	271	N	90383	271
1	6	272	N	90384	272
1	6	273	N	90385	273
1	6	274	N	90386	274
1	6	275	N	90387	275
1	6	276	N	90388	276
1	6	277	N	90389	277
1	6	278	N	90390	278
1	6	279	N	90391	279
1	6	280	N	90392	280
1	6	281	N	90393	281
1	6	282	N	90394	282
1	6	283	N	90395	283
1	6	284	N	90396	284
1	6	285	N	90397	285
1	6	286	N	90398	286

2	1	1	Y	4097	1
2	1	2	Y	4098	2
2	1	3	Y	4099	3
2	1	4	Y	4100	4
2	1	5	Y	4101	5
2	1	6	Y	4102	6
2	1	7	Y	4103	7
2	1	8	Y	4104	8
2	1	9	Y	4105	9
2	1	138	N	135306	138
2	1	139	N	135307	139
2	1	140	N	135308	140
2	1	141	N	135309	141
2	1	142	N	135310	142
2	1	143	N	135311	143
2	1	144	N	135312	144
2	1	145	N	135313	145
2	1	146	N	135314	146
2	1	147	N	135315	147
2	1	148	N	135316	148
2	1	149	N	135317	149
2	1	150	N	135318	150
2	1	151	N	135319	151
2	1	152	N	135320	152

2	1	153	N	135321	153
2	1	154	N	135322	154
2	1	155	N	135323	155
2	1	156	N	135324	156
2	1	157	N	135325	157
2	1	158	N	135326	158
2	2	65	Y	8257	65
2	2	66	Y	8258	66
2	2	67	Y	8259	67
2	2	68	Y	8260	68
2	2	69	Y	8261	69
2	2	70	Y	8262	70
2	2	71	Y	8263	71
2	2	72	Y	8264	72
2	2	73	Y	8265	73
2	2	202	N	139466	202
2	2	203	N	139467	203
2	2	204	N	139468	204
2	2	205	N	139469	205
2	2	206	N	139470	206
2	2	207	N	139471	207
2	2	208	N	139472	208
2	2	209	N	139473	209
2	2	210	N	139474	210

2	2	211	N	139475	211
2	2	212	N	139476	212
2	2	213	N	139477	213
2	2	214	N	139478	214
2	2	215	N	139479	215
2	2	216	N	139480	216
2	2	217	N	139481	217
2	2	218	N	139482	218
2	2	219	N	139483	219
2	2	220	N	139484	220
2	2	221	N	139485	221
2	2	222	N	139486	222
2	3	129	Y	12417	129
2	3	130	Y	12418	130
2	3	131	Y	12419	131
2	3	132	Y	12420	132
2	3	133	Y	12421	133
2	3	134	Y	12422	134
2	3	135	Y	12423	135
2	3	136	Y	12424	136
2	3	137	Y	12425	137
2	3	10	N	143370	10
2	3	11	N	143371	11
2	3	12	N	143372	12

2	3	13	N	143373	13
2	3	14	N	143374	14
2	3	15	N	143375	15
2	3	16	N	143376	16
2	3	17	N	143377	17
2	3	18	N	143378	18
2	3	19	N	143379	19
2	3	20	N	143380	20
2	3	21	N	143381	21
2	3	22	N	143382	22
2	3	23	N	143383	23
2	3	24	N	143384	24
2	3	25	N	143385	25
2	3	26	N	143386	26
2	3	27	N	143387	27
2	3	28	N	143388	28
2	3	29	N	143389	29
2	3	30	N	143390	30
2	4	193	Y	16577	193
2	4	194	Y	16578	194
2	4	195	Y	16579	195
2	4	196	Y	16580	196
2	4	197	Y	16581	197
2	4	198	Y	16582	198

2	4	199	Y	16583	199
2	4	200	Y	16584	200
2	4	201	Y	16585	201
2	4	74	N	147530	74
2	4	75	N	147531	75
2	4	76	N	147532	76
2	4	77	N	147533	77
2	4	78	N	147534	78
2	4	79	N	147535	79
2	4	80	N	147536	80
2	4	81	N	147537	81
2	4	82	N	147538	82
2	4	83	N	147539	83
2	4	84	N	147540	84
2	4	85	N	147541	85
2	4	86	N	147542	86
2	4	87	N	147543	87
2	4	88	N	147544	88
2	4	89	N	147545	89
2	4	90	N	147546	90
2	4	91	N	147547	91
2	4	92	N	147548	92
2	4	93	N	147549	93
2	4	94	N	147550	94



2	5	257	Y	20737	257
2	5	258	Y	20738	258
2	5	259	Y	20739	259
2	5	260	Y	20740	260
2	5	261	Y	20741	261
2	5	262	Y	20742	262
2	5	263	Y	20743	263
2	5	264	Y	20744	264
2	5	265	Y	20745	265
2	5	394	N	151946	394
2	5	395	N	151947	395
2	5	396	N	151948	396
2	5	397	N	151949	397
2	5	398	N	151950	398
2	5	399	N	151951	399
2	5	400	N	151952	400
2	5	401	N	151953	401
2	5	402	N	151954	402
2	5	403	N	151955	403
2	5	404	N	151956	404
2	5	405	N	151957	405
2	5	406	N	151958	406
2	5	407	N	151959	407
2	5	408	N	151960	408

2	5	409	N	151961	409
2	5	410	N	151962	410
2	5	411	N	151963	411
2	5	412	N	151964	412
2	5	413	N	151965	413
2	5	414	N	151966	414
2	6	321	Y	24897	321
2	6	322	Y	24898	322
2	6	323	Y	24899	323
2	6	324	Y	24900	324
2	6	325	Y	24901	325
2	6	326	Y	24902	326
2	6	327	Y	24903	327
2	6	328	Y	24904	328
2	6	329	Y	24905	329
2	6	458	N	156106	458
2	6	459	N	156107	459
2	6	460	N	156108	460
2	6	461	N	156109	461
2	6	462	N	156110	462
2	6	463	N	156111	463
2	6	464	N	156112	464
2	6	465	N	156113	465
2	6	466	N	156114	466

2	6	467	N	156115	467
2	6	468	N	156116	468
2	6	469	N	156117	469
2	6	470	N	156118	470
2	6	471	N	156119	471
2	6	472	N	156120	472
2	6	473	N	156121	473
2	6	474	N	156122	474
2	6	475	N	156123	475
2	6	476	N	156124	476
2	6	477	N	156125	477
2	6	478	N	156126	478
3	1	1	Y	4097	1
3	1	2	Y	4098	2
3	1	3	Y	4099	3
3	1	4	Y	4100	4
3	1	5	Y	4101	5
3	1	6	Y	4102	6
3	1	7	Y	4103	7
3	1	8	Y	4104	8
3	1	9	Y	4105	9
3	1	202	N	200906	202
3	1	203	N	200907	203
3	1	204	N	200908	204

3	1	205	N	200909	205
3	1	206	N	200910	206
3	1	207	N	200911	207
3	1	208	N	200912	208
3	1	209	N	200913	209
3	1	210	N	200914	210
3	1	211	N	200915	211
3	1	212	N	200916	212
3	1	213	N	200917	213
3	1	214	N	200918	214
3	1	215	N	200919	215
3	1	216	N	200920	216
3	1	217	N	200921	217
3	1	218	N	200922	218
3	1	219	N	200923	219
3	1	220	N	200924	220
3	1	221	N	200925	221
3	1	222	N	200926	222
3	2	65	Y	8257	65
3	2	66	Y	8258	66
3	2	67	Y	8259	67
3	2	68	Y	8260	68
3	2	69	Y	8261	69
3	2	70	Y	8262	70

3	2	71	Y	8263	71
3	2	72	Y	8264	72
3	2	73	Y	8265	73
3	2	138	N	204938	138
3	2	139	N	204939	139
3	2	140	N	204940	140
3	2	141	N	204941	141
3	2	142	N	204942	142
3	2	143	N	204943	143
3	2	144	N	204944	144
3	2	145	N	204945	145
3	2	146	N	204946	146
3	2	147	N	204947	147
3	2	148	N	204948	148
3	2	149	N	204949	149
3	2	150	N	204950	150
3	2	151	N	204951	151
3	2	152	N	204952	152
3	2	153	N	204953	153
3	2	154	N	204954	154
3	2	155	N	204955	155
3	2	156	N	204956	156
3	2	157	N	204957	157
3	2	158	N	204958	158

3	3	129	Y	12417	129
3	3	130	Y	12418	130
3	3	131	Y	12419	131
3	3	132	Y	12420	132
3	3	133	Y	12421	133
3	3	134	Y	12422	134
3	3	135	Y	12423	135
3	3	136	Y	12424	136
3	3	137	Y	12425	137
3	3	74	N	208970	74
3	3	75	N	208971	75
3	3	76	N	208972	76
3	3	77	N	208973	77
3	3	78	N	208974	78
3	3	79	N	208975	79
3	3	80	N	208976	80
3	3	81	N	208977	81
3	3	82	N	208978	82
3	3	83	N	208979	83
3	3	84	N	208980	84
3	3	85	N	208981	85
3	3	86	N	208982	86
3	3	87	N	208983	87
3	3	88	N	208984	88

3	3	89	N	208985	89
3	3	90	N	208986	90
3	3	91	N	208987	91
3	3	92	N	208988	92
3	3	93	N	208989	93
3	3	94	N	208990	94
3	4	193	Y	16577	193
3	4	194	Y	16578	194
3	4	195	Y	16579	195
3	4	196	Y	16580	196
3	4	197	Y	16581	197
3	4	198	Y	16582	198
3	4	199	Y	16583	199
3	4	200	Y	16584	200
3	4	201	Y	16585	201
3	4	10	N	213002	10
3	4	11	N	213003	11
3	4	12	N	213004	12
3	4	13	N	213005	13
3	4	14	N	213006	14
3	4	15	N	213007	15
3	4	16	N	213008	16
3	4	17	N	213009	17
3	4	18	N	213010	18

3	4	19	N	213011	19
3	4	20	N	213012	20
3	4	21	N	213013	21
3	4	22	N	213014	22
3	4	23	N	213015	23
3	4	24	N	213016	24
3	4	25	N	213017	25
3	4	26	N	213018	26
3	4	27	N	213019	27
3	4	28	N	213020	28
3	4	29	N	213021	29
3	4	30	N	213022	30
3	5	257	Y	20737	257
3	5	258	Y	20738	258
3	5	259	Y	20739	259
3	5	260	Y	20740	260
3	5	261	Y	20741	261
3	5	262	Y	20742	262
3	5	263	Y	20743	263
3	5	264	Y	20744	264
3	5	265	Y	20745	265
3	5	458	N	217546	458
3	5	459	N	217547	459
3	5	460	N	217548	460



3	5	461	N	217549	461
3	5	462	N	217550	462
3	5	463	N	217551	463
3	5	464	N	217552	464
3	5	465	N	217553	465
3	5	466	N	217554	466
3	5	467	N	217555	467
3	5	468	N	217556	468
3	5	469	N	217557	469
3	5	470	N	217558	470
3	5	471	N	217559	471
3	5	472	N	217560	472
3	5	473	N	217561	473
3	5	474	N	217562	474
3	5	475	N	217563	475
3	5	476	N	217564	476
3	5	477	N	217565	477
3	5	478	N	217566	478
3	6	321	Y	24897	321
3	6	322	Y	24898	322
3	6	323	Y	24899	323
3	6	324	Y	24900	324
3	6	325	Y	24901	325
3	6	326	Y	24902	326

3	6	327	Y	24903	327
3	6	328	Y	24904	328
3	6	329	Y	24905	329
3	6	394	N	221578	394
3	6	395	N	221579	395
3	6	396	N	221580	396
3	6	397	N	221581	397
3	6	398	N	221582	398
3	6	399	N	221583	399
3	6	400	N	221584	400
3	6	401	N	221585	401
3	6	402	N	221586	402
3	6	403	N	221587	403
3	6	404	N	221588	404
3	6	405	N	221589	405
3	6	406	N	221590	406
3	6	407	N	221591	407
3	6	408	N	221592	408
3	6	409	N	221593	409
3	6	410	N	221594	410
3	6	411	N	221595	411
3	6	412	N	221596	412
3	6	413	N	221597	413
3	6	414	N	221598	414

4	1	1	Y	4097	1
4	1	2	Y	4098	2
4	1	3	Y	4099	3
4	1	4	Y	4100	4
4	1	5	Y	4101	5
4	1	6	Y	4102	6
4	1	7	Y	4103	7
4	1	8	Y	4104	8
4	1	9	Y	4105	9
4	1	266	N	266506	266
4	1	267	N	266507	267
4	1	268	N	266508	268
4	1	269	N	266509	269
4	1	270	N	266510	270
4	1	271	N	266511	271
4	1	272	N	266512	272
4	1	273	N	266513	273
4	1	274	N	266514	274
4	1	275	N	266515	275
4	1	276	N	266516	276
4	1	277	N	266517	277
4	1	278	N	266518	278
4	1	279	N	266519	279
4	1	280	N	266520	280

4	1	281	N	266521	281
4	1	282	N	266522	282
4	1	283	N	266523	283
4	1	284	N	266524	284
4	1	285	N	266525	285
4	1	286	N	266526	286
4	2	65	Y	8257	65
4	2	66	Y	8258	66
4	2	67	Y	8259	67
4	2	68	Y	8260	68
4	2	69	Y	8261	69
4	2	70	Y	8262	70
4	2	71	Y	8263	71
4	2	72	Y	8264	72
4	2	73	Y	8265	73
4	2	330	N	270666	330
4	2	331	N	270667	331
4	2	332	N	270668	332
4	2	333	N	270669	333
4	2	334	N	270670	334
4	2	335	N	270671	335
4	2	336	N	270672	336
4	2	337	N	270673	337
4	2	338	N	270674	338

4	2	339	N	270675	339
4	2	340	N	270676	340
4	2	341	N	270677	341
4	2	342	N	270678	342
4	2	343	N	270679	343
4	2	344	N	270680	344
4	2	345	N	270681	345
4	2	346	N	270682	346
4	2	347	N	270683	347
4	2	348	N	270684	348
4	2	349	N	270685	349
4	2	350	N	270686	350
4	3	129	Y	12417	129
4	3	130	Y	12418	130
4	3	131	Y	12419	131
4	3	132	Y	12420	132
4	3	133	Y	12421	133
4	3	134	Y	12422	134
4	3	135	Y	12423	135
4	3	136	Y	12424	136
4	3	137	Y	12425	137
4	3	394	N	274826	394
4	3	395	N	274827	395
4	3	396	N	274828	396

4	3	397	N	274829	397
4	3	398	N	274830	398
4	3	399	N	274831	399
4	3	400	N	274832	400
4	3	401	N	274833	401
4	3	402	N	274834	402
4	3	403	N	274835	403
4	3	404	N	274836	404
4	3	405	N	274837	405
4	3	406	N	274838	406
4	3	407	N	274839	407
4	3	408	N	274840	408
4	3	409	N	274841	409
4	3	410	N	274842	410
4	3	411	N	274843	411
4	3	412	N	274844	412
4	3	413	N	274845	413
4	3	414	N	274846	414
4	4	193	Y	16577	193
4	4	194	Y	16578	194
4	4	195	Y	16579	195
4	4	196	Y	16580	196
4	4	197	Y	16581	197
4	4	198	Y	16582	198

4	4	199	Y	16583	199
4	4	200	Y	16584	200
4	4	201	Y	16585	201
4	4	458	N	278986	458
4	4	459	N	278987	459
4	4	460	N	278988	460
4	4	461	N	278989	461
4	4	462	N	278990	462
4	4	463	N	278991	463
4	4	464	N	278992	464
4	4	465	N	278993	465
4	4	466	N	278994	466
4	4	467	N	278995	467
4	4	468	N	278996	468
4	4	469	N	278997	469
4	4	470	N	278998	470
4	4	471	N	278999	471
4	4	472	N	279000	472
4	4	473	N	279001	473
4	4	474	N	279002	474
4	4	475	N	279003	475
4	4	476	N	279004	476
4	4	477	N	279005	477
4	4	478	N	279006	478

4	5	257	Y	20737	257
4	5	258	Y	20738	258
4	5	259	Y	20739	259
4	5	260	Y	20740	260
4	5	261	Y	20741	261
4	5	262	Y	20742	262
4	5	263	Y	20743	263
4	5	264	Y	20744	264
4	5	265	Y	20745	265
4	5	10	N	282634	10
4	5	11	N	282635	11
4	5	12	N	282636	12
4	5	13	N	282637	13
4	5	14	N	282638	14
4	5	15	N	282639	15
4	5	16	N	282640	16
4	5	17	N	282641	17
4	5	18	N	282642	18
4	5	19	N	282643	19
4	5	20	N	282644	20
4	5	21	N	282645	21
4	5	22	N	282646	22
4	5	23	N	282647	23
4	5	24	N	282648	24



4	5	25	N	282649	25
4	5	26	N	282650	26
4	5	27	N	282651	27
4	5	28	N	282652	28
4	5	29	N	282653	29
4	5	30	N	282654	30
4	6	321	Y	24897	321
4	6	322	Y	24898	322
4	6	323	Y	24899	323
4	6	324	Y	24900	324
4	6	325	Y	24901	325
4	6	326	Y	24902	326
4	6	327	Y	24903	327
4	6	328	Y	24904	328
4	6	329	Y	24905	329
4	6	74	N	286794	74
4	6	75	N	286795	75
4	6	76	N	286796	76
4	6	77	N	286797	77
4	6	78	N	286798	78
4	6	79	N	286799	79
4	6	80	N	286800	80
4	6	81	N	286801	81
4	6	82	N	286802	82

4	6	83	N	286803	83
4	6	84	N	286804	84
4	6	85	N	286805	85
4	6	86	N	286806	86
4	6	87	N	286807	87
4	6	88	N	286808	88
4	6	89	N	286809	89
4	6	90	N	286810	90
4	6	91	N	286811	91
4	6	92	N	286812	92
4	6	93	N	286813	93
4	6	94	N	286814	94
5	1	1	Y	4097	1
5	1	2	Y	4098	2
5	1	3	Y	4099	3
5	1	4	Y	4100	4
5	1	5	Y	4101	5
5	1	6	Y	4102	6
5	1	7	Y	4103	7
5	1	8	Y	4104	8
5	1	9	Y	4105	9
5	1	330	N	332106	330
5	1	331	N	332107	331
5	1	332	N	332108	332

5	1	333	N	332109	333
5	1	334	N	332110	334
5	1	335	N	332111	335
5	1	336	N	332112	336
5	1	337	N	332113	337
5	1	338	N	332114	338
5	1	339	N	332115	339
5	1	340	N	332116	340
5	1	341	N	332117	341
5	1	342	N	332118	342
5	1	343	N	332119	343
5	1	344	N	332120	344
5	1	345	N	332121	345
5	1	346	N	332122	346
5	1	347	N	332123	347
5	1	348	N	332124	348
5	1	349	N	332125	349
5	1	350	N	332126	350
5	2	65	Y	8257	65
5	2	66	Y	8258	66
5	2	67	Y	8259	67
5	2	68	Y	8260	68
5	2	69	Y	8261	69
5	2	70	Y	8262	70

5	2	71	Y	8263	71
5	2	72	Y	8264	72
5	2	73	Y	8265	73
5	2	266	N	336138	266
5	2	267	N	336139	267
5	2	268	N	336140	268
5	2	269	N	336141	269
5	2	270	N	336142	270
5	2	271	N	336143	271
5	2	272	N	336144	272
5	2	273	N	336145	273
5	2	274	N	336146	274
5	2	275	N	336147	275
5	2	276	N	336148	276
5	2	277	N	336149	277
5	2	278	N	336150	278
5	2	279	N	336151	279
5	2	280	N	336152	280
5	2	281	N	336153	281
5	2	282	N	336154	282
5	2	283	N	336155	283
5	2	284	N	336156	284
5	2	285	N	336157	285
5	2	286	N	336158	286

5	3	129	Y	12417	129
5	3	130	Y	12418	130
5	3	131	Y	12419	131
5	3	132	Y	12420	132
5	3	133	Y	12421	133
5	3	134	Y	12422	134
5	3	135	Y	12423	135
5	3	136	Y	12424	136
5	3	137	Y	12425	137
5	3	458	N	340426	458
5	3	459	N	340427	459
5	3	460	N	340428	460
5	3	461	N	340429	461
5	3	462	N	340430	462
5	3	463	N	340431	463
5	3	464	N	340432	464
5	3	465	N	340433	465
5	3	466	N	340434	466
5	3	467	N	340435	467
5	3	468	N	340436	468
5	3	469	N	340437	469
5	3	470	N	340438	470
5	3	471	N	340439	471
5	3	472	N	340440	472

5	3	473	N	340441	473
5	3	474	N	340442	474
5	3	475	N	340443	475
5	3	476	N	340444	476
5	3	477	N	340445	477
5	3	478	N	340446	478
5	4	193	Y	16577	193
5	4	194	Y	16578	194
5	4	195	Y	16579	195
5	4	196	Y	16580	196
5	4	197	Y	16581	197
5	4	198	Y	16582	198
5	4	199	Y	16583	199
5	4	200	Y	16584	200
5	4	201	Y	16585	201
5	4	394	N	344458	394
5	4	395	N	344459	395
5	4	396	N	344460	396
5	4	397	N	344461	397
5	4	398	N	344462	398
5	4	399	N	344463	399
5	4	400	N	344464	400
5	4	401	N	344465	401
5	4	402	N	344466	402

5	4	403	N	344467	403
5	4	404	N	344468	404
5	4	405	N	344469	405
5	4	406	N	344470	406
5	4	407	N	344471	407
5	4	408	N	344472	408
5	4	409	N	344473	409
5	4	410	N	344474	410
5	4	411	N	344475	411
5	4	412	N	344476	412
5	4	413	N	344477	413
5	4	414	N	344478	414
5	5	257	Y	20737	257
5	5	258	Y	20738	258
5	5	259	Y	20739	259
5	5	260	Y	20740	260
5	5	261	Y	20741	261
5	5	262	Y	20742	262
5	5	263	Y	20743	263
5	5	264	Y	20744	264
5	5	265	Y	20745	265
5	5	74	N	348234	74
5	5	75	N	348235	75
5	5	76	N	348236	76

5	5	77	N	348237	77
5	5	78	N	348238	78
5	5	79	N	348239	79
5	5	80	N	348240	80
5	5	81	N	348241	81
5	5	82	N	348242	82
5	5	83	N	348243	83
5	5	84	N	348244	84
5	5	85	N	348245	85
5	5	86	N	348246	86
5	5	87	N	348247	87
5	5	88	N	348248	88
5	5	89	N	348249	89
5	5	90	N	348250	90
5	5	91	N	348251	91
5	5	92	N	348252	92
5	5	93	N	348253	93
5	5	94	N	348254	94
5	6	321	Y	24897	321
5	6	322	Y	24898	322
5	6	323	Y	24899	323
5	6	324	Y	24900	324
5	6	325	Y	24901	325
5	6	326	Y	24902	326



5	6	327	Y	24903	327
5	6	328	Y	24904	328
5	6	329	Y	24905	329
5	6	10	N	352266	10
5	6	11	N	352267	11
5	6	12	N	352268	12
5	6	13	N	352269	13
5	6	14	N	352270	14
5	6	15	N	352271	15
5	6	16	N	352272	16
5	6	17	N	352273	17
5	6	18	N	352274	18
5	6	19	N	352275	19
5	6	20	N	352276	20
5	6	21	N	352277	21
5	6	22	N	352278	22
5	6	23	N	352279	23
5	6	24	N	352280	24
5	6	25	N	352281	25
5	6	26	N	352282	26
5	6	27	N	352283	27
5	6	28	N	352284	28
5	6	29	N	352285	29
5	6	30	N	352286	30

6	1	1	Y	4097	1
6	1	2	Y	4098	2
6	1	3	Y	4099	3
6	1	4	Y	4100	4
6	1	5	Y	4101	5
6	1	6	Y	4102	6
6	1	7	Y	4103	7
6	1	8	Y	4104	8
6	1	9	Y	4105	9
6	1	394	N	397706	394
6	1	395	N	397707	395
6	1	396	N	397708	396
6	1	397	N	397709	397
6	1	398	N	397710	398
6	1	399	N	397711	399
6	1	400	N	397712	400
6	1	401	N	397713	401
6	1	402	N	397714	402
6	1	403	N	397715	403
6	1	404	N	397716	404
6	1	405	N	397717	405
6	1	406	N	397718	406
6	1	407	N	397719	407
6	1	408	N	397720	408

6	1	409	N	397721	409
6	1	410	N	397722	410
6	1	411	N	397723	411
6	1	412	N	397724	412
6	1	413	N	397725	413
6	1	414	N	397726	414
6	2	65	Y	8257	65
6	2	66	Y	8258	66
6	2	67	Y	8259	67
6	2	68	Y	8260	68
6	2	69	Y	8261	69
6	2	70	Y	8262	70
6	2	71	Y	8263	71
6	2	72	Y	8264	72
6	2	73	Y	8265	73
6	2	458	N	401866	458
6	2	459	N	401867	459
6	2	460	N	401868	460
6	2	461	N	401869	461
6	2	462	N	401870	462
6	2	463	N	401871	463
6	2	464	N	401872	464
6	2	465	N	401873	465
6	2	466	N	401874	466

6	2	467	N	401875	467
6	2	468	N	401876	468
6	2	469	N	401877	469
6	2	470	N	401878	470
6	2	471	N	401879	471
6	2	472	N	401880	472
6	2	473	N	401881	473
6	2	474	N	401882	474
6	2	475	N	401883	475
6	2	476	N	401884	476
6	2	477	N	401885	477
6	2	478	N	401886	478
6	3	129	Y	12417	129
6	3	130	Y	12418	130
6	3	131	Y	12419	131
6	3	132	Y	12420	132
6	3	133	Y	12421	133
6	3	134	Y	12422	134
6	3	135	Y	12423	135
6	3	136	Y	12424	136
6	3	137	Y	12425	137
6	3	266	N	405770	266
6	3	267	N	405771	267
6	3	268	N	405772	268

6	3	269	N	405773	269
6	3	270	N	405774	270
6	3	271	N	405775	271
6	3	272	N	405776	272
6	3	273	N	405777	273
6	3	274	N	405778	274
6	3	275	N	405779	275
6	3	276	N	405780	276
6	3	277	N	405781	277
6	3	278	N	405782	278
6	3	279	N	405783	279
6	3	280	N	405784	280
6	3	281	N	405785	281
6	3	282	N	405786	282
6	3	283	N	405787	283
6	3	284	N	405788	284
6	3	285	N	405789	285
6	3	286	N	405790	286
6	4	193	Y	16577	193
6	4	194	Y	16578	194
6	4	195	Y	16579	195
6	4	196	Y	16580	196
6	4	197	Y	16581	197
6	4	198	Y	16582	198

6	4	199	Y	16583	199
6	4	200	Y	16584	200
6	4	201	Y	16585	201
6	4	330	N	409930	330
6	4	331	N	409931	331
6	4	332	N	409932	332
6	4	333	N	409933	333
6	4	334	N	409934	334
6	4	335	N	409935	335
6	4	336	N	409936	336
6	4	337	N	409937	337
6	4	338	N	409938	338
6	4	339	N	409939	339
6	4	340	N	409940	340
6	4	341	N	409941	341
6	4	342	N	409942	342
6	4	343	N	409943	343
6	4	344	N	409944	344
6	4	345	N	409945	345
6	4	346	N	409946	346
6	4	347	N	409947	347
6	4	348	N	409948	348
6	4	349	N	409949	349
6	4	350	N	409950	350

6	5	257	Y	20737	257
6	5	258	Y	20738	258
6	5	259	Y	20739	259
6	5	260	Y	20740	260
6	5	261	Y	20741	261
6	5	262	Y	20742	262
6	5	263	Y	20743	263
6	5	264	Y	20744	264
6	5	265	Y	20745	265
6	5	138	N	413834	138
6	5	139	N	413835	139
6	5	140	N	413836	140
6	5	141	N	413837	141
6	5	142	N	413838	142
6	5	143	N	413839	143
6	5	144	N	413840	144
6	5	145	N	413841	145
6	5	146	N	413842	146
6	5	147	N	413843	147
6	5	148	N	413844	148
6	5	149	N	413845	149
6	5	150	N	413846	150
6	5	151	N	413847	151
6	5	152	N	413848	152

6	5	153	N	413849	153
6	5	154	N	413850	154
6	5	155	N	413851	155
6	5	156	N	413852	156
6	5	157	N	413853	157
6	5	158	N	413854	158
6	6	321	Y	24897	321
6	6	322	Y	24898	322
6	6	323	Y	24899	323
6	6	324	Y	24900	324
6	6	325	Y	24901	325
6	6	326	Y	24902	326
6	6	327	Y	24903	327
6	6	328	Y	24904	328
6	6	329	Y	24905	329
6	6	202	N	417994	202
6	6	203	N	417995	203
6	6	204	N	417996	204
6	6	205	N	417997	205
6	6	206	N	417998	206
6	6	207	N	417999	207
6	6	208	N	418000	208
6	6	209	N	418001	209
6	6	210	N	418002	210



6	6	211	N	418003	211
6	6	212	N	418004	212
6	6	213	N	418005	213
6	6	214	N	418006	214
6	6	215	N	418007	215
6	6	216	N	418008	216
6	6	217	N	418009	217
6	6	218	N	418010	218
6	6	219	N	418011	219
6	6	220	N	418012	220
6	6	221	N	418013	221
6	6	222	N	418014	222

Table 3: Example Derivation Results

## Authors' Addresses

Jordan Head (editor)  
Juniper Networks  
1137 Innovation Way  
Sunnyvale, CA  
United States of America  
Email: jhead@juniper.net

Tony Przygienda  
Juniper Networks  
1137 Innovation Way  
Sunnyvale, CA  
United States of America  
Email: prz@juniper.net

Wen Lin  
Juniper Networks  
10 Technology Park Drive  
Westford, MA  
United States of America  
Email: wlin@juniper.net