Guide to SDN with OpenConfig for M&E

Introduction

Bandwidth management should be seriously considered in multi-switch, dense, uncompressed ST2110 M&E applications. Flows should be "load balanced" over multiple physical bearers, and / or across multiple spine devices, in environments with large flow sizes, and the expectation of high flow density. In these types of environments, some form of orchestration is highly recommended.

Arista's MCS (Media Control Service) is a valid option, offering a robust middleware solution that can be integrated with the Broadcast Controller above. MCS provides the route finding / stitching and bandwidth management functions, and abstracts these into a simple northbound API, with the addition of rich streaming telemetry to provide flow visibility and network topology.

An alternative is for a vendor to create their own orchestration / SDN layer, performing their own pathfinding and bandwidth management, leveraging Arista API's. This approach allows maximum flexibility, and enables the vendor to add as much secret sauce to add value and differentiation as desired.

This document will focus on the technical aspects of using Openconfig as the API used to communicate with an underlying Arista network to provide SDN services.

Network / Host Connection Models

Arista is flexible with the way switch topologies are built, and how hosts are connected. A strong recommendation would be to use standard layer 3 routed connectivity, leveraging ECMP, for inter-switch connectivity. You can use the standard Arista L3 UCN guide to drive choices for the unicast base config.

The rest of this document will assume that multi-switch topologies - hub and spoke, leaf and spine for example, will be connected using layer 3, routed ports.

Hosts can be connected either with switchports, or routed ports (as shown in Figure 1) - we can deliver to routed ports using static multicast routing, and to switchports, using static snooping entries.

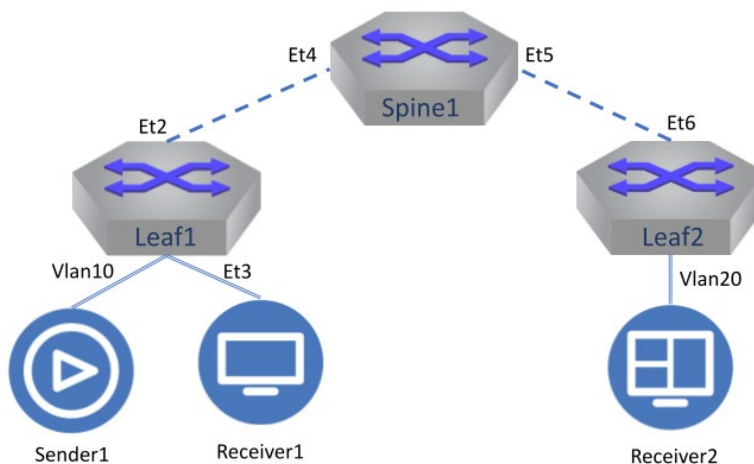

Figure 1: L2 and L3 Senders and Receivers

Switch Config Prerequisites

The configuration examples here assume the CLI changes are mandatory from EOS version 4.23 onwards. For more info about the differences, please see this field notice.

Enabling management interfaces

Enabling gNMI:

The following CLI will enable the switch to receive and act on gNMI (gRPC Network Management Interface) connections. By default, any management or front panel port connection that provides the appropriate reachability can be used.

```
management api gnmi
   transport grpc default
```

If the management connection is in non-default VRF, then we have to specify the vrf name in this config as shown below:

```
management api gnmi
   transport grpc default
   vrf <vrfName>
```

Enabling Static Multicast

For each layer 3 interface where static multicast is planned to be used - whether that is a routed host, routed up/downlink, or SVI (for L2 attached hosts) - it should be enabled per interface basis, in order to ensure that the static routed multicast is forwarded. Below, you can see examples for the different use-cases: (Note for clarity, other config, such as PTP has been omitted)

Routed host, or routed up/downlink:

```
interface Ethernet52/1
   description media-spine-1
   no switchport
   ip address 172.24.125.14/30
   multicast ipv4 static
```

SVI supporting L2 attached hosts:

```
interface Vlan2252
   ip address 172.24.225.17/28
   multicast ipv4 static
```

OpenConfig

Basics

OpenConfig (https://openconfig.net) describes a "consistent set of vendor-neutral data models (written in YANG) based on actual operational needs from use cases and requirements from multiple network operators". These data models are then used for both device configuration, status, and streaming telemetry operations. You can find more info on the openconfig models here.

Arista uses these models where possible, however, when official models are not available, and customers need extended capabilities, Arista can support its own private models. Static multicast routing and snooping capabilities fall into this category.

The OpenConfig model approach does not mandate the communication transport method, and a strength is indeed the ability for vendors and implementers to choose from a variety of transport options.

We recommend using gNMI , which provides a secure, connected, high performance transport, and can be used for configuration, status collection and streaming telemetry. The gNMI spec can be found here.

An easy to use command line app for gNMI can be found here, and this allows for simple and quick prototyping of configuration, monitoring and telemetry streaming. There are various libraries that will allow you to simply integrate gNMI programmatically, this python one for example https://github.com/aristanetworks/pyopenconfig/tree/master/pyopenconfig.

For the examples in this document, the gNMI command line app described earlier is used.

So, what can we do with OpenConfig using gNMI?

The basic constructs are:

- **Capabilities.** Enables the gNMI client to determine the set of models that are supported by the target.
- **Get.** Used by a client to retrieve the current snapshot of the status of the path accessed
- **Set.** Umbrella term to do Delete/Replace/Update operation
    - **Delete.** Used by a client to remove configuration sections
    - **Replace.** Used by a client to fully replace config sections with new contents
    - **Update.** Used to add new content to a config section (Note Update does not rewrite the config section, it only appends to the existing config section. If there is a need to completely rewrite the config section, Replace or Delete+Update must be used)
- **Subscribe.** Used by a client to subscribe to real-time streaming telemetry.

Recommendations for High Performance:

When building your own Bandwidth Management Orchestration solution, to achieve High performance output following are the recommendations,

- Build your own gNMI client rather than using pre-built gNMI clients.
    - Pre-built gNMI clients are great while starting with OpenConfig but for a production ready scalable system it is recommended to build your own client using a high performance programming language
- Group as many operations as possible within a single gNMI transaction
    - Each transaction will incur protocol related transportation costs. So this can be worked around if the Orchestration solution can group as many operations as possible within a single gNMI transaction
- Initialise the gNMI connection once and keep it open as long as possible for subsequent transactions.
    - Every new gNMI connection has to go through AAA authentication. Arista switches cache the AAA sessions so subsequent transactions using the same connection and username/password do not need to go through AAA authentication again.
- Pick a programming language that allows parallel (or near-parallel) programming. This provides a huge performance boost if performing multi-switch operations where hundreds of route updates can be pushed in a fraction of the time as opposed to serially programming every switch one at a time.

Look out for:

As, within a single gNMI transaction we can perform multiple operations, care must be taken to validate syntax and the existence of an entity on the switch before pushing the config. If there are any syntax errors, the whole transaction will fail.

If there are any errors in the route path update (missing entity fields), the transaction is silently accepted with possible undesired changes pushed to the config. So caution must be taken to make sure the entity that needs updating is present before pushing the config.

For example, this happens when we have missing IIF/OIF or invalid interfaces for a stream that doesn't exist on the switch. OpenConfig does not validate the data itself, but just the required schema. Responsibility for validating the data lies within the Orchestration level.

Consider the following update , where OIF is a routed port Ethernet54. But Leaf1 does not have Ethernet54 and has only Ethernet54/1. This update is accepted and configured on the switch with no validation.

```
# gnmi -addr Leaf1:6030 -username admin -password admin replace "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]" '{
    "key-g": "239.1.1.1/32",
    "key-s": "10.10.10.1/32",
    "iif": "Ethernet1",
    "toCpu": false,
    "routePriority": 255,
    "oifs": [{
          "index": "Ethernet54",
          "value": false
    }]
}'
```

Possible error scenario of the above example, is when we try to create a route and the update field does not have IIF or OIF populated, the corresponding field on the switch will look empty, thereby appearing like an orphaned/dangling flow. In the following example, a dangling static route entry is created.

```
# gnmi -addr Leaf1:6030 -username admin -password admin replace "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]" '{
    "key-g": "239.1.1.1/32",
    "key-s": "10.10.10.1/32",
    "iif": "Ethernet1",
    "toCpu": false,
    "routePriority": 255
}'
```

gNMI app command:

Using the gnmi app as an example, here is the syntax of a replace command:

```
gnmi -addr Leaf1:6030 -username admin -password admin replace "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=237.1.1.1/32][key-s=10.10.10.10/32]" '{
    "key-g": "237.1.1.1/32",
    "key-s": "10.10.10.10/32",
    "iif": "Ethernet1",
    "toCpu": false,
    "routePriority": 255,
    "oifs": [{
     "index": "Ethernet2",
     "value": false
    }, {
     "index": "Ethernet3",
     "value": false
    }, {
     "index": "Ethernet4",
     "value": false
    }]
}'
```

Dissecting the full command, it can be split into three sections - command, path and contents

Command: In this case, it's a ***replace***.

`gnmi -addr  Leaf1:6030 -username admin -password admin replace`

Path: This is the path for setting a static multicast route, in the default VRF, for the (S,G) = (10,10,10,10/32, 237.1.1.1/32).

```
"/arista/eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=237.1.1.1/32][key-s=10.10.10.10/32]"
```

Contents: Now we describe the contents for the static routing instruction for the (S,G) described in the path:

```
'{"key-g": "237.1.1.1/32", "key-s": "10.10.10.10/32", "iif": "Ethernet1", "toCpu":
false, "routePriority": 255, "oifs": [{"index": "Ethernet2", "value": false},{"index":
"Ethernet3", "value": false},{"index": "Ethernet4", "value": false}]}'
```

The example describes how to configure a static route from "Ethernet1" - the iif, to be forwarded to "Ethernet2, 3 and 4" - the oifs.

Examples of manipulating static paths going through the fabric

L3 Routed Path Management

L3 routed connections would be used for:

- Directing flows from routed source ports to routed uplinks
- Directing flows from routed downlinks to routed destination ports
- Directing flows from source SVI's to routed uplinks
- Directing flows from routed downlinks to destination SVI's
- Directing flows from (local) source SVI's to destination SVI's

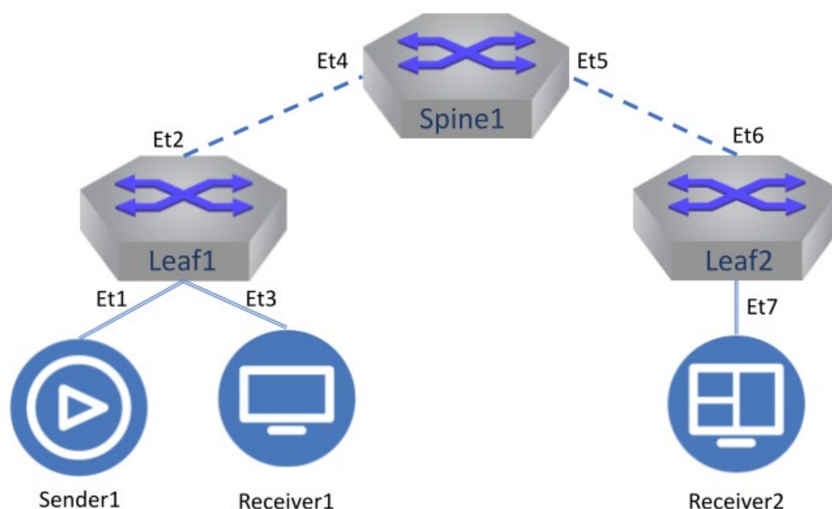

Figure 2: L3 Senders and Receivers

To try a few sample workflows on Figure2 topology, that cover the above connections,

1. Create a new route from Sender1 to Receiver1 (S,G = 10.10.10.1/32, 239.1.1.1)

Routes will go from routed ports Ethernet1 to Ethernet3. We'd suggest using the 'replace' attribute as this will create the new entry, and replace anything that was previously there.

```
# gnmi -addr Leaf1:6030 -username admin -password admin replace "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]" '{
    "key-g": "239.1.1.1/32",
    "key-s": "10.10.10.1/32",
    "iif": "Ethernet1",
    "toCpu": false,
    "routePriority": 255,
    "oifs": [{
     "index": "Ethernet3",
     "value": false
    }]
}'
```

Note: While constructing static multicast routes routePriority needs to be set to 255 which gives this configuration the highest priority.

2. Create a new route from Sender1 to Receiver2 (S,G = 10.10.10.1/32, 239.1.1.1)

This would require sending 3 gNMI requests to 3 different devices (Leaf1, Spine, Leaf2). Orchestration layer can choose to optimise the On leaf1, the route will go from routed ports Ethernet1 to Ethernet2. Here we will use the **'update'** attribute, as it will update our existing ip route without impacting the existing route to Ethernet3.

```
# gnmi -addr Leaf1:6030 -username admin -password admin update "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]" '{
    "key-g": "239.1.1.1/32",
    "key-s": "10.10.10.1/32",
    "iif": "Ethernet1",
    "toCpu": false,
    "routePriority": 255,
    "oifs": [{
          "index": "Ethernet2",
          "value": false
    }]
}'
```

On Spine, the route will go from routed ports Ethernet4 to Ethernet5. Here we can use the **'replace'** attribute as it is a new desired route.

```
# gnmi -addr Spine:6030 -username admin -password admin replace "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]" '{
    "key-g": "239.1.1.1/32",
    "key-s": "10.10.10.1/32",
    "iif": "Ethernet4",
    "toCpu": false,
    "routePriority": 255,
    "oifs": [{
          "index": "Ethernet5",
          "value": false
    }]
}'
```

On Leaf2, the route will go from routed ports Ethernet6 to Ethernet7 connecting the receiver. Here we can use the **'replace'** attribute as it is a new desired route.

```
# gnmi -addr Leaf2:6030 -username admin -password admin replace "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]" '{
    "key-g": "239.1.1.1/32",
    "key-s": "10.10.10.1/32",
    "iif": "Ethernet6",
    "toCpu": false,
    "routePriority": 255,
    "oifs": [{
          "index": "Ethernet7",
          "value": false
    }]
}'
```

3. To Delete route to Receiver1

Action required is to delete the oif to Ethernet3 on Leaf1. This will make sure the stream to Receiver2 on Leaf2 is untouched. Use only 'delete' operation here and not 'replace'. Though 'replace' will converge to the desired end state of retaining the uplink Ethernet port Ethernet2, it will momentarily remove the entire config including the route to Ethernet2 and then add Ethernet2 back . This will cause traffic disruption.

```
# gnmi -addr Leaf1:6030 -username admin -password admin delete "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]/oifs[index=Ethernet3]"
```

Delete route to Receiver2

Action required is to delete routes on all three devices, Leaf1, Spine and Leaf2. Deletion of the entire route can be done as follows.

```
gnmi -addr Leaf1:6030 -username admin -password admin delete "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]"
```

Similar to setting routes on 3 devices, the above delete should also be issued to the three devices.

Topology with Source and Destination SVIs:

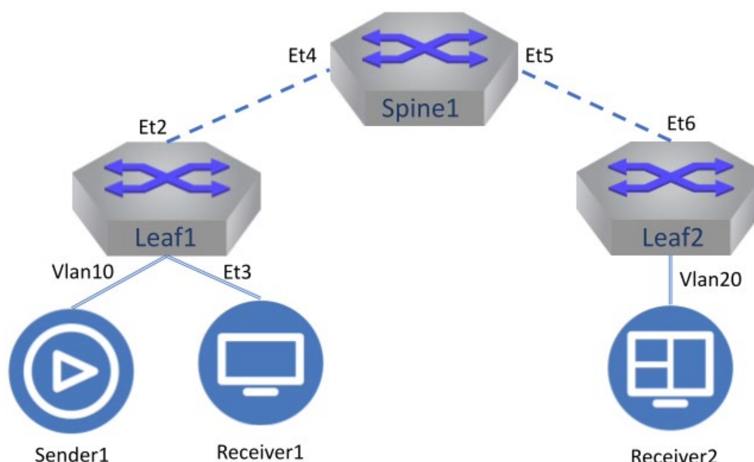

Figure 1: L2 and L3 Senders and Receivers

In this new case instead of using routed Ethernet ports in iif/oif SVI are used in respective fields.

1. Route on Leaf1 from Source SVI Vlan10 connected to Sender1 to routed port Ethernet3 connected to Receiver1

```
# gnmi -addr Leaf1:6030 -username admin -password admin replace "/arista/
eos/routing/multicast/routeconfig/static/vrfConfig[vrfName=default]/
staticMcastRoute[key-g=239.1.1.1/32][key-s=10.10.10.1/32]" '{
    "key-g": "239.1.1.1/32",
    "key-s": "10.10.10.1/32",
    "iif": "Vlan10",
    "toCpu": false,
    "routePriority": 255,
    "oifs": [{
     "index": "Ethernet3",
     "value": false
    }]
}'
```

L2 Destination Path Management

L2 static snooping entries would be used for:

- Directing flows from SVI's to L2 hosts(receivers) in that SVI

Assuming the flow is either sourced in the target SVI, or has been statically routed to the SVI already, then static snooping can be used to deliver to the L2 attached host. For example, if Ethernet8 is configured as part of Vlan200 and is routed as a multicast receiver, then a snooping entry for 237.1.1.1 is added to Vlan200 for interface Ethernet8

```
gnmi -addr Leaf1:6030 -username "admin" -password "admin" replace "/arista/eos/bridging/
igmpsnooping/config/vlanConfig[vlanId=200]/ipGroup[addr=237.1.1.1]" '{
    "addr": "237.1.1.1",
    "intf": [{
     "index": "Ethernet8",
     "value": true
    }]
}'
```

Other considerations

Ingress bandwidth Policing

Typically, in order to ensure that the Network Controller can perform path-finding and bandwidth management effectively, it's important that the Network Controller can be very sure what bandwidth is being used, and is available, in the network.

A typical approach is to manage / police traffic at ingress, so that the Network Controller can now be confident that it understands all the (significant) traffic in the network.

Approaches will vary, but can include:

- Denying all ingress multicast traffic, except known / desired flows
- Applying IGMP filtering on L2 interfaces to prevent unexpected IGMP joins
- Preventing dynamic multicast routing by not configuring PIM

Policing traffic can be done with OpenConfig. Following sections explore the commands to do the same.

Creating ACL using OpenConfig gNMI

Create ACL based policer per flow

```
gnmi -addr Leaf1:6030 -username "admin" -password "admin" update "/acl/acl-sets" '{
    "acl-set": [{
     "acl-entries": {
            "acl-entry": [{
                   "actions": {
                          "config": {
                                 "forwarding-action": "ACCEPT",
                                 "log-action": "LOG_NONE"
                          }
                   },
                   "config": {
                          "sequence-id": 10
                   },
                   "ipv4": {
                          "config": {
                                 "destination-address": "239.0.0.32/32",
                                 "source-address": "172.1.1.32/32"
```

```
                           }
                    },
                    "sequence-id": 10
             }]
      },
      "config": {
             "name": "172_1_1_32-239_0_0_32",
             "type": "ACL_IPV4"
      },
      "name": "172_1_1_32-239_0_0_32",
      "type": "ACL_IPV4"
     }]
}'
```

To Delete

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" delete /acl/acl-sets/acl-set[name=172_1_1_32-239_0_0_32][type=ACL_IPV4]
```

Creating Class Maps using OpenConfig gNMI

Match the above created ACL in class map

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" update "/arista/eos/qos/acl/input/cli/cmapType[type=mapQos]" '{
  "arista-exp-eos-qos-acl-config:cmap": [
    {
    "match": [
           {
           "option": "matchIpAccessGroup",
           "strValue": "172_1_1_32-239_0_0_32"
           }
    ],
    "matchCondition": "matchConditionAny",
    "name": "172_1_1_32-239_0_0_32"
    }],
  "arista-exp-eos-qos-acl-config:type": "mapQos"
}
```

To Delete

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" delete /arista/eos/qos/acl/input/cli/cmapType[type=mapQos]/cmap[name=172_1_1_32-239_0_0_32]
```

Creating Policy Maps using OpenConfig gNMI

Create a policy map per ingress interface per flow. This can constitute multiple flows

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" update "/arista/eos/qos/acl/
input/cli/pmapType[type=mapQos]" '
{
  "arista-exp-eos-qos-acl-config:pmap": [
    {
    "classAction": [
          {
          "name": "172_1_1_32-239_1_1_32",
          "policer": {
          "bc": 993,
          "bcUnit": "burstUnitKBytes",
          "cir": "8",
          "cirUnit": "rateUnitMbps",
          "cmdVersion": 2,
          "redActions": [
          {
                "actionType": "actionSetDrop"
          }
          ]
          }
          }
    ],
    "classActionDefault": {
          "name": "class-default",
          "policer": {
          "cmdVersion": 1
          }
    },
    "classDefault": {
          "match": [
          {
          "option": "matchIpAccessGroup"
          }
          ]
    },
    "name": "Ethernet15"
    }
  ],
  "arista-exp-eos-qos-acl-config:type": "mapQos"
}'
```

To Delete

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" delete /
arista/eos/qos/acl/input/cli/pmapType[type=mapQos]/pmap[name=Ethernet15]/
classAction[name=172_1_1_32-239_1_1_32]
```

This would remove just the class map of this particular stream but retain the policy-map config for further stream's class-map to be added. So to clear out the policy-map config completely:

Deleting the entire policer

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" delete /arista/eos/qos/acl/
input/cli/pmapType[type=mapQos]/pmap[name=Ethernet15]
```

Creating and attaching Service Policy to source Ethernet interface using OpenConfig gNMI

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" update "/arista/eos/qos/
input/config/cli/servicePolicyConfig[key-direction=input][key-pmapName=Ethernet15][key-
type=mapQos]" '{
  "arista-exp-eos-qos-config:intfIds": [
      {
      "index": "Ethernet15",
      "value": true
      }
  ],
  "arista-exp-eos-qos-config:key-direction": "input",
  "arista-exp-eos-qos-config:key-pmapName": "Ethernet15",
  "arista-exp-eos-qos-config:key-type": "mapQos"
}'
```

To Delete

```
gnmi -addr Leaf1:6030 -username "admin" -password "arista" delete /arista/eos/qos/input/
config/cli/servicePolicyConfig[key-direction=input][key-pmapName=Ethernet15][key-type=mapQos]
```

Interoperation with dynamic multicast

This section is not specific to static multicast configured via openconfig - it's true for all static and dynamic multicast systems, but worth including here nonetheless.

It is not possible to use a mixture of static and dynamic multicast routing operations on the same (S,G). Once an (S,G) has been routed using a static entry, this (S,G) will no longer be available to be routed using dynamic PIM techniques. It is therefore recommended that if dynamic multicast routing via PIM is a requirement in a system where static multicasts are also being stitched (via OpenConfig or even MCS), that static groups encompass one range of multicast groups, and dynamic multicast encompasses another - and there shall be no overlap.

EOS Documentation

OpenConfig Configuration Guide
https://aristanetworks.github.io/openmgmt/configuration/openconfig/

Arista OpenConfig Supported Paths
https://eos.arista.com/path-report/

Santa Clara—Corporate Headquarters
5453 Great America Parkway,
Santa Clara, CA 95054

Phone: +1-408-547-5500
Fax: +1-408-538-8920
Email: info@arista.com

Ireland—International Headquarters
3130 Atlantic Avenue
Westpark Business Campus
Shannon, Co. Clare
Ireland

Vancouver—R&D Office
9200 Glenlyon Pkwy, Unit 300
Burnaby, British Columbia
Canada V5J 5J8

San Francisco—R&D and Sales Office
1390 Market Street, Suite 800
San Francisco, CA 94102

India—R&D Office
Global Tech Park, Tower A, 11th Floor
Marathahalli Outer Ring Road
Devarabeesanahalli Village, Varthur Hobli
Bangalore, India 560103

Singapore—APAC Administrative Office
9 Temasek Boulevard
#29-01, Suntec Tower Two
Singapore 038989

Nashua—R&D Office
10 Tara Boulevard
Nashua, NH 03062

Copyright © 2022 Arista Networks, Inc. All rights reserved. CloudVision, and EOS are registered trademarks and Arista Networks is a trademark of Arista Networks, Inc. All other company names are trademarks of their respective holders. Information in this document is subject to change without notice. Certain features may not yet be available. Arista Networks, Inc. assumes no responsibility for any errors that may appear in this document. April 19, 2022 02-0101-01