Configuring Generic UDP Encapsulation

Read this section to get an overview and know how to configure the Generic UDP Encapsulation.

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
Outer IP Header-Driven Hash Computation for Incoming GUE Packets	Release 7.11.1	We now offer you the flexibility of using only the outer IP header to calculate the hashing for incoming Generic UDP Encapsulation (GUE) packets. On enabling this feature, only the outer IP source and destination addresses are used for hashing calculations. The inner IP addresses are not considered, providing a simpler method of distribution. Previously, both inner IP and outer IP headers were used for ECMP hashing the incoming GUE packets. The feature introduces these changes: **CLI:** • **hw-module profile gue underlay-hash** **YANG Data Models:** • New XPaths for `Cisco-IOS-XR-npu-hw-profile-cfg.yang` (see GitHub, YANG Data Models Navigator) The command is supported on Q200-based ASICs.

Feature Name	Release Information	Feature Description
Generic UDP Encapsulation	Release 7.3.1	This feature enables you to add an additional header to packets to identify or authenticate the data using UDP. Encapsulating packets in UDP leverages the use of the UDP source port to provide entropy to Equal Cost Multi-Path (ECMP) hashing. It provides significant performance benefits for load-balancing. This command is introduced for this feature: **decapsulate gue**

• Understand Generic UDP Encapsulation, on page 2
• Flexible Assignment of UDP Port Numbers for Decapsulation, on page 9

Understand Generic UDP Encapsulation

UDP encapsulation is a technique of adding network headers to packets and then encapsulating the packets within the User Datagram Protocol (UDP).

Encapsulating packets using UDP facilitates efficient transport across networks. By leveraging Receive Side Scaling (RSS) and Equal Cost Multipath (ECMP) routing, UDP provides significant performance benefits for load-balancing. The use of the UDP source port provides entropy to ECMP hashing and provides the ability to use the IP source or destination, and the L4 Port for load-balancing entropy.

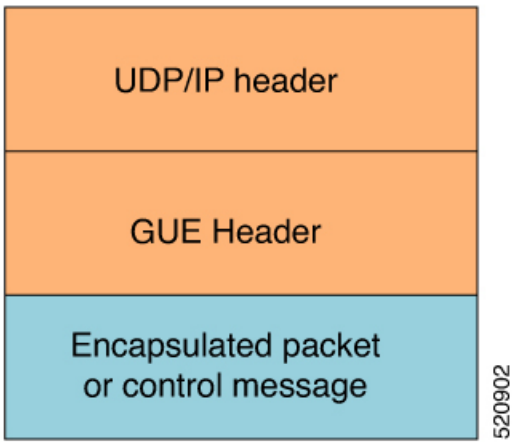
Traditional mechanisms like Generic Routing Encapsulation (GRE) can handle only the outer Source IP address and parts of the destination address. They may not provide sufficient load balancing entropy.

Generic UDP Encapsulation (GUE) is a UDP-based network encapsulation protocol that encapsulates IPv4 and IPv6 packets. GUE provides native UDP encapsulation and defines an additional header, which helps to determine the payload carried by the IP packet. The additional header can include items, such as a virtual networking identifier, security data for validating or authenticating the GUE header, congestion control data, and so on.

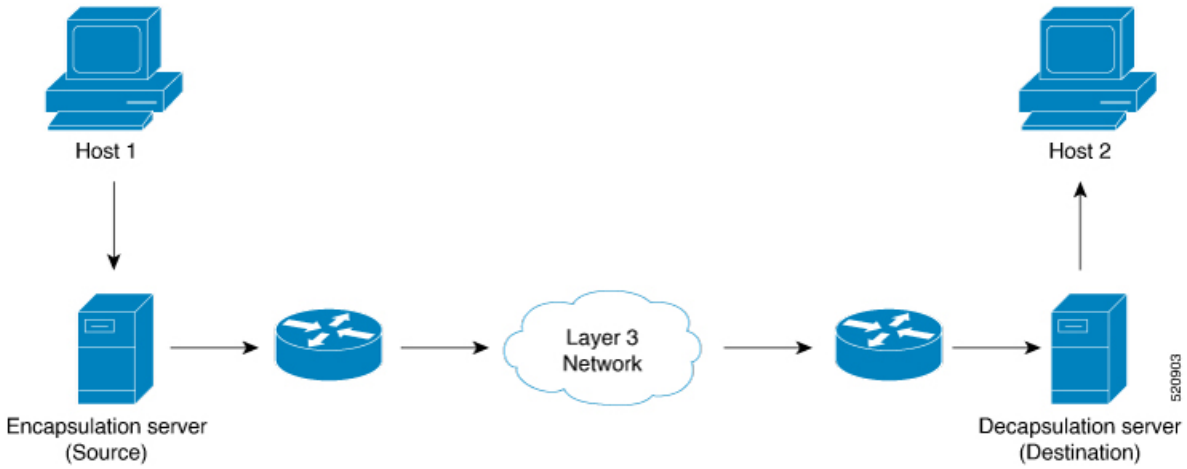
In GUE, the payload is encapsulated in an IP packet that can be IPv4 or IPv6 Carrier. The UDP header is added to provide extra hashing parameters, and optional payload demultiplexing. At the decapsulation node, the Carrier IP and UDP headers are removed, and the packet is forwarded based on the inner payload.

A GUE packet has the general format:

Figure 1: GUE Packet Format



For example, in this scenario, if the data stream is sent from Host 1 to Host 2. The server acts as a GUE encapsulator that sends the packets from Host 1. The server, on the other end receiving the data, validates the data for the valid carrier IP and UDP header and decapsulates the data.



GUE has various variants, but variant 1 of GUE allows direct encapsulation of IPv4 and IPv6 in UDP. This technique saves encapsulation overhead on links for the use of IP encapsulation, and also need not allocate a separate UDP port number for IP-over-UDP encapsulation.

Variant 1 has no GUE header, but a UDP packet carries an IP packet. The first two bits of the UDP payload is the GUE variant field and match with the first two bits of the version number in the IP header.

Benefits of using GUE

- Allows direct encapsulation of payloads, such as IPv4 and IPv6 in the UDP packet.
  - You can use UDP port for demultiplexing payloads.
  - You can use a single UDP port, allowing systems to employ parsing models to identify payloads.
- Leverages the UDP header for entropy labels by encoding a tuple-based source port.

- Leverages source IP addresses for load-balance encoding. The destination too could be terminated based on a subnet providing additional bits for entropy.
- Avoids special handling for transit nodes because they only see an IP-UDP packet with some payload..
- Eases implementation of UDP tunneling with GUE. This is because of the direct encapsulation method of the payloads into UDP.

Restrictions

- Supports Generic UDP Decapsulation for only variant 1.
- Receives IPv4 packets with the defined GUE port of 6080.
- Decapsulates IPv6 packets with the defined GUE port of 6080.
- Receives MPLS packets with the UDPoMPLS port of 6635.
- Range of source or destination ports is not supported.
- Range, Source, or Destination addresses are not supported, but subnet mask enteries are allowed.
- To perform decapsulation, a destination Port is mandatory.
- Terminating GRE after GUE or GUE after GRE is not supported.
- Terminating a label such as a VPN Deaggregation after GUE termination is not supported.
- Slow path support is not supported. To resolve the inner IP Adjacency, use the **cef proactive-arp-nd enable** command.
- Running the **clear all** command doesn't clear the interface of all its existing configurations.

Note To use only outer IP header (L3 and L4) for calculating the hashing for incoming GUE packets, use the **hw-module profile gue underlay-hash enable** command. Otherwise, by default, both outer IP header (L3 and L4) and inner IP header (L3 and L4) are considered for calculating the hashing for incoming GUE packets.

The **hw-module profile gue underlay-hash enable** command is currently not supported on the P100-based and Q100-based ASICs.

Configure GUE

Configuring GUE

Use the following configuration workflow to configure GUE:

1. Configure separate GUE decap tunnel UDP destination port numbers for IPv4, IPv6, and MPLS using **hw-module profile gue udp-dest-port** command.
2. Configure a traffic class: Create a traffic class and specify various criteria for classifying packets using the match commands, and an instruction on how to evaluate these match commands.
3. Configure a policy map: Define a policy map and associate the traffic class with the traffic policy.

4. Apply the policy for each VRF, and apply this policy on all the interfaces that are part of the VRF.

Configuration Example for GUE IPv4

1. Configure separate UDP port numbers for IPv4, IPv6, and MPLS using **hw-module profile gue udp-dest-port**command.

```
Router# configure
Router# hw-module profile gue udp-dest-port ipv4 6080 ipv6 6080 mpls 6635

Router# commit
```

Note While adding or removing the **hw-module profile gue udp-dest-port** command, you must reload the router.

2. Configure a traffic class:

```
Router# configure
Router(config)# class-map type traffic match-all udp-v4
Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.255
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.255
Router(config-cmap)# match protocol udp
Router(config-cmap)# match destination-port 6080
Router(config-cmap)# end-class-map
Router(config)# commit

Router(config)# class-map type traffic match-all udp-mpls1
Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.255
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.255
Router(config-cmap)# match protocol udp
Router(config-cmap)# match destination-port 6635
Router(config-cmap)# end-class-map
Router(config)# commit

Router(config)# class-map type traffic match-all udp-v6
Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.255
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.255
Router(config-cmap)# match protocol udp
Router(config-cmap)# match destination-port 6080
Router(config-cmap)# end-class-map
Router(config)# commit
```

3. Define a policy map, and associate the traffic class with the traffic policy:

```
Router(config)# policy-map type pbr magic-decap

Router(config-pmap)# class type traffic udp-v4
Router(config-pmap-c)# decapsulate gue variant 1
Router(config-pmap-c)# exit

Router(config-pmap)# class type traffic udp-v6
Router(config-pmap-c)# decapsulate gue variant 1
Router(config-pmap-c)# exit

Router(config-pmap)# class type traffic udp-mpls1
Router(config-pmap-c)# decapsulate gue variant 1
Router(config-pmap-c)# exit
```

```
Router(config-pmap)# class type traffic class-default
Router(config-pmap-c)# exit
Router(config-pmap)# end-policy-map
Router(config)# commit
Router(config)# exit
```

4. Apply the policy for each VRF:

```
Router# configure
Router(config)# vrf-policy
Router(config-vrf-policy)# vrf default address-family ipv4 policy type pbr input magic-decap
Router(config-vrf-policy)# commit
```

Running Configuration:

```
class-map type traffic match-all udp-v4
 match destination-address ipv4 220.100.20.0 255.255.255.255
 match source-address ipv4 210.100.20.0 255.255.255.255
 match protocol udp
 match destination-port 6080
 end-class-map
!
class-map type traffic match-all udp-v6
 match destination-address ipv4 220.100.20.0 255.255.255.255
 match source-address ipv4 210.100.20.0 255.255.255.255
 match protocol udp
 match destination-port 6080
 end-class-map
!
class-map type traffic match-all udp-mpls1
 match destination-address ipv4 220.100.20.0 255.255.255.255
 match source-address ipv4 210.100.20.0 255.255.255.255
 match protocol udp
 match destination-port 6635
 end-class-map
!
policy-map type pbr magic-decap
 class type traffic udp-v4
  decapsulate gue variant 1
 !
 class type traffic udp-v6
  decapsulate gue variant 1
 !
 class type traffic udp-mpls1
  decapsulate gue variant 1
 !
 class type traffic class-default
 !
 end-policy-map
!


vrf-policy
 vrf default address-family ipv4 policy type pbr input magic-decap
!
```

Verification

To view the set of counter values accumulated for the packets that match the class-map:

```
Router# show policy-map type pbr addr-family ipv4 statistics

VRF Name:       default
Policy-Name:    pmap
```

```
Policy Type:    pbr
Addr Family:    IPv4

Class:     cmap-loop1
     Classification statistics      (packets/bytes)
       Matched             :              0/0
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :              0/0

Class:     cmap-loop6
     Classification statistics      (packets/bytes)
       Matched             :              0/0
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :              0/0

Class:     cmap-loop2
     Classification statistics      (packets/bytes)
       Matched             :              0/0
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :              0/0

Class:     cmap-loop3
     Classification statistics      (packets/bytes)
       Matched             :       198325306/17849277540
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :       198325306/17849277540

Class:     cmap-loop4
    Classification statistics      (packets/bytes)
       Matched             :              0/0
     Transmitted statistics         (packets/bytes)
       Total Transmitted   :              0/0
```

To clear the policy-map counters for each class-map rule, use the **clear vrf** command:

```
Router# clear vrf default address-family ipv4 statistics
```

Outer IP Header-Driven Hash Computation for Incoming GUE Packets

When multiple paths with the same cost are available for forwarding traffic, ECMP hashing is used to determine the path to select for each packet. Each packet that needs to be forwarded is processed using a hashing algorithm. The hashing algorithm considers specific packet fields such as source IP, destination IP, source port, and destination port, and generates a hash value. The generated hash value is then mapped to one of the available paths. The selected path is used to forward the packet to its destination. The goal is to distribute the traffic evenly across the available paths to prevent congestion and utilize the network resources efficiently.

Now you can use only the outer IP header (L3 and L4) for calculating the hash value for incoming GUE packets and completely ignore the usage of the inner IP header. This functionality is configurable using the CLI command **hw-module profile gue underlay-hash**. This is supported for both GUE termination (decapsulation) and GUE transit (pass-through) nodes. By default, the feature is disabled; that is, both outer IP header (L3 and L4) and inner IP header (L3 and L4) are used for calculating the hashing for GUE packets.

Benefits

- **Load Balancing Efficiency:** By hashing only on the outer IP and L4 information, the packets with the same source and destination IP addresses and L4 ports consistently follow the same path in a load-balanced environment. This helps maintain session affinity or stickiness, as the inner IP addresses or L4 port numbers may change dynamically within the encapsulated packets.

- **Network Security:** Ignoring the inner IP helps preserve privacy and confidentiality within the encapsulated packets. By focusing on the outer IP and L4 headers, the network device does not have visibility into the inner IP addressing scheme or the specific content encapsulated within the packet, which enhances security.
- **Network Scalability:** Ignoring the inner IP reduces the complexity and overhead of packet processing, improving overall network performance and scalability, especially in high-throughput environments.

Configure Outer IP Header-Driven Hash Computation for Incoming GUE Packets

This section describes how to configure hashing with only outer IP for GUE packets.

Configuration Example

Use the following configuration to enable hashing with only outer IP for GUE packets:

```
Router# configure
Router# hw-module profile gue underlay-hash enable
Router# commit
```

Running Configuration

```
RP/0/RP0/CPU0:ios(config)#show running-config
hw-module profile gue underlay-hash enable
end
```

Verification

Following is the show command output before enabling hashing with only outer IP for GUE packets.

```
RP/0/RP0/CPU0:ios#show dpa objects sys location 0/RP0/CPU0  | include gue
  uint32_t gue_ipv4_port => 0
  uint32_t gue_ipv6_port => 0
  uint32_t gue_mpls_port => 0
  ofa_bool_t gue_underlay_hash => FALSE
```

Following is the show command output after enabling hashing with only outer IP for GUE packets.

```
RP/0/RP0/CPU0:ios#show dpa objects sys location 0/RP0/CPU0 | include gue
  uint32_t gue_ipv4_port => 0
  uint32_t gue_ipv6_port => 0
  uint32_t gue_mpls_port => 0
  ofa_bool_t gue_underlay_hash => TRUE
```

Flexible Assignment of UDP Port Numbers for Decapsulation

Table 2: Feature History Table

Feature Name	Release Information	Feature Description
Flexible Assignment of UDP Port Numbers for Decapsulation	Release 7.3.3	This feature gives you the flexibility to assign UDP port numbers from 1000 through 6400, through which IPv4, IPv6, and MPLS packets can be decapsulated. Such flexibility allows you to segregate the ingress traffic based on a QoS policy. In earlier releases, you could assign only default ports for decapsulation. The following command is introduced for this feature: `hw-module profile gue udp-dest-port ipv4 <port number> ipv6 <port number> mpls <port number>`

This feature provides decapsulation support for GUE packets. In GUE, the payload is encapsulated in an IP packet—IPv4 or IPv6 carrier. The UDP header is added to provide extra hashing parameters and optional payload demultiplexing. At the decapsulation node, the carrier IP and UDP headers are removed, and the packet is forwarded based on the inner payload. Prior to Release 7.3.3, packets were decapsulated using UDP port numbers 6080, 6615, and 6635 for IPv4, IPv6, and MPLS payloads respectively. Starting from Release 7.3.3, you can assign UDP port numbers from 1000 through 64000 to decapsulate IPv4, IPv6, and MPLS packets. Define different port numbers for IPv4, IPv6, and MPLS.

Guidelines for Setting up Decapsulation Using Flexible Port Numbers

Apply these guidelines while assigning flexible port numbers for decapsulation:

Packet	IPv4	IPv6	MPLS
UDP Outer Header	Configure IPv4 port on the hardware module.	Configure IPv6 port on the hardware module.	Configure MPLS port on the hardware module.
Encapsulation Outer Header	Configure an IPv4 encapsulation outer header that matches with the class map source.		
Inner Payload	Note that packets are forwarded based on the inner IPv4 payload.	Note that packets are forwarded based on the inner IPv6 payload.	Note that packets are forwarded based on the inner MPLS payload.

Note

- During the decapsulation of the IPv4, IPv6, and MPLS packets, the following headers are removed:
  - The UDP outer header
  - The IPv4 encapsulation outer header
- Select different values for each of these protocols. Valid port numbers are from 1000 through 64000.

Restrictions

The following restrictions are applicable while configuring unique GUE destination port numbers to decapsulate IPv4, IPv6, and MPLS packets using UDP:

- While configuring the tunnel, select one of the following:
  - Match only 16 unique source IP addresses as shown in the example:

    ```
    Router(config-cmap)#match source-address ipv4 210.100.20.0 255.255.255.255
    ```

  - Match a combination of 64 unique source and destination IP addresses as shown in the example:

    ```
    Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.255
    Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.255
    ```

- The Classless Inter-Domain Routing (CIDR) value in the source IP address subnet mask must be only /32.
- The destination address subnet mask supports all CIDR values. However, the destination address along with the subnet mask must be unique for all the three UDP payload types—IPv4, IPv6, and MPLS. The configuration fails when the destination IP address and the subnet mask are the same for all three payloads as seen in this example:

```
Router(config)#class-map type traffic match-all SRTE-GUE-DECAP-IPv4
Router(config-cmap)#match destination-address ipv4 10.216.101.0 255.255.255.255
..
Router(config)#class-map type traffic match-all SRTE-GUE-DECAP-IPv6
Router(config-cmap)#match destination-address ipv4 10.216.101.0 255.255.255.255
..
Router(config)#class-map type traffic match-all SRTE-GUE-DECAP-MPLS
Router(config-cmap)#match destination-address ipv4 10.216.101.0 255.255.255.255
..
```

Configuring Port Numbers for Decapsulation

By configuring different port numbers on the destination router, you can match and direct traffic to different paths. For example, traffic for a specific video service can be decapsulated and sent through different ports. The steps that are involved in configuring port numbers for decapsulation are:

1. Configure the UDP destination ports for decapsulation of the required payloads.
2. Configure the traffic class to match the ports.
3. Define a policy map, and associate the traffic class with the traffic policy.

4. Apply the policy for each VRF.

Note For the hardware module flexible port configuration to take effect you must reload the line card.

Configuration Example

```
Hw-module configuration:
====================
Router# configure
Router# hw-module profile gue udp-dest-port ipv4 1001 ipv6 1002 mpls 1003

Class-map configuration:
====================
Router# configure
Router(config)# class-map type traffic match-all udp-v4
Router(config-cmap)# match protocol udp
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.255
Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.255
Router(config-cmap)# match destination-port 1001
Router(config-cmap)# end-class-map
Router(config)# commit

Router(config)# class-map type traffic match-all udp-v6
Router(config-cmap)# match protocol udp
Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.255
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.255
Router(config-cmap)# match destination-port 1002
Router(config-cmap)# end-class-map
Router(config)# commit

Router(config)# class-map type traffic match-all udp-mpls1
Router(config-cmap)# match protocol udp
Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.255
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.255
Router(config-cmap)# match destination-port 1003
Router(config-cmap)# end-class-map
Router(config)# commit

Ingress Policy-map configuration:
=========================
Router(config)# policy-map type pbr magic-decap
Router(config-pmap)# class type traffic udp-v4
Router(config-pmap-c)# decapsulate gue variant 1
Router(config-pmap-c)# exit

Router(config-pmap)# class type traffic udp-v6
Router(config-pmap-c)# decapsulate gue variant 1
Router(config-pmap-c)# exit

Router(config-pmap)# class type traffic udp-mpls1
Router(config-pmap-c)# decapsulate gue variant 1
Router(config-pmap-c)# exit

Router(config-pmap)# class type traffic class-default
Router(config-pmap-c)# exit
Router(config-pmap)# end-policy-map
Router(config)# commit
Router(config)# exit
```

```
Applying policy per VRF:
=======================================
Router# configure
Router(config)# vrf-policy
Router(config-vrf-policy)# vrf default address-family ipv4 policy type pbr input magic-decap
Router(config-vrf-policy)# commit
```

Running Configuration

```
!! File saved at 16:01:32 UTC Mon Feb 07 2022 by cisco
!! IOS XR Configuration 7.3.3.10I
!! Last configuration change at Mon Feb  7 15:35:11 2022 by cisco
!
logging console disable
username cisco
 group root-lr
 group cisco-support
 secret 10
$6$gHKmE1YZAo71BE1.$3KYogrvOdJxTRPZgYPGXUXkO4PqQMr2E6oYvJO4ngBmuaGsF2nAB/m1NP5Il3zh9HTzBI/k4r8PwWSbsARsmp.
!
vrf vrf-gre
 address-family ipv4 unicast
 !
 address-family ipv6 unicast
 !
!
line console
 exec-timeout 0 0
 absolute-timeout 0
 session-timeout 0
!
line default
 exec-timeout 0 0
 absolute-timeout 0
 session-timeout 0
!
!arp vrf default 29.0.1.2 0000.1122.2929 ARPA
call-home
 service active
 contact smart-licensing
 profile CiscoTAC-1
  active
  destination transport-method http
 !
!
ipv6 access-list abf6-gre
 1 permit ipv6 any any nexthop1 ipv6 201:0:1::2
!
ipv4 access-list abf-gre
 1 permit ipv4 any any nexthop1 ipv4 201.0.1.2
!
class-map type traffic match-all udp-v4
 match destination-address ipv4 220.100.20.0 255.255.255.255
 match source-address ipv4 210.100.20.0 255.255.255.255
 match protocol udp
 match destination-port 1001
 end-class-map
!
class-map type traffic match-all udp-v6
 match destination-address ipv4 220.100.20.0 255.255.255.255
 match source-address ipv4 210.100.20.0 255.255.255.255
 match protocol udp
```

```
 match destination-port 1002
 end-class-map
!
class-map type traffic match-all udp-mpls1
 match destination-address ipv4 220.100.20.0 255.255.255.255
 match source-address ipv4 210.100.20.0 255.255.255.255
 match protocol udp
 match destination-port 1003
 end-class-map
!
policy-map type pbr pbr-gre
 class type traffic class-default
  redirect ipv4 nexthop 202.0.1.2
 !
 end-policy-map
!
policy-map type pbr magic-decap
 class type traffic udp-v4
  decapsulate gue variant 1
 !
 class type traffic udp-v6
  decapsulate gue variant 1
 !
 class type traffic udp-mpls1
  decapsulate gue variant 1
 !
 class type traffic class-default
 !
 end-policy-map
!
interface Bundle-Ether25
 ipv4 address 25.0.1.1 255.255.255.0
 ipv6 address 25:0:1::1/64
 ipv6 enable
 shutdown
!
interface Bundle-Ether28
 ipv4 address 28.0.1.1 255.255.255.0
!
interface Loopback0
 ipv4 address 10.10.10.1 255.255.255.255
!
<output truncated>
interface MgmtEth0/RP0/CPU0/0
 ipv4 address dhcp
!
interface MgmtEth0/RP1/CPU0/0
 ipv4 address dhcp
!
interface BVI23
 ipv4 address 23.0.1.1 255.255.255.0
 ipv6 address 23:0:1::1/64
 ipv6 enable
 shutdown
!
interface BVI29
 ipv4 address 29.0.1.1 255.255.255.0
 ipv6 enable
 shutdown
!
interface HundredGigE0/0/0/0
 shutdown
!
```

```
<output truncated>
 l2transport
 !
!
interface HundredGigE0/0/0/24
 service-policy type pbr input pbr-gre
 ipv4 address 24.0.1.1 255.255.255.0
 ipv6 address 24:0:1::1/64
 ipv6 enable
!
interface HundredGigE0/0/0/24.24
 ipv4 address 24.0.24.1 255.255.255.0
 ipv6 enable
 encapsulation dot1q 24
!
interface HundredGigE0/0/0/25
 bundle id 25 mode on
!
interface HundredGigE0/0/0/26
 ipv4 address 26.0.1.1 255.255.255.0
 ipv6 address 26:0:1::1/64
 ipv6 enable
!
interface HundredGigE0/0/0/27
 ipv4 address 27.0.1.1 255.255.255.0
 ipv6 enable
!
interface HundredGigE0/0/0/27.27
 ipv4 address 27.0.27.1 255.255.255.0
 ipv6 address 27:0:27::1/64
 ipv6 enable
 shutdown
 encapsulation dot1q 27
!
interface HundredGigE0/0/0/28
 bundle id 28 mode active
!
interface HundredGigE0/0/0/29
 ipv4 address 29.0.1.1 255.255.255.0
 ipv6 enable
!
<output truncated>
interface HundredGigE0/1/0/24
 ipv4 address 124.0.1.1 255.255.255.0
 ipv6 address 124:0:1::1/64
 ipv6 enable
!
<output truncated>
!
interface HundredGigE0/1/0/30
 bundle id 28 mode active
!
interface HundredGigE0/1/0/31
 ipv4 address 31.0.1.1 255.255.255.0
 ipv6 address 31:0:1::1/64
 shutdown
!
<output truncated>
!
route-policy pass
  pass
end-policy
!
router static
```

```
 address-family ipv4 unicast
  201.0.1.0/24 tunnel-ip1
  201.0.1.0/24 tunnel-ip2
  201.0.1.0/24 tunnel-ip3
  201.0.1.0/24 tunnel-ip4
 !
 address-family ipv6 unicast
  201:0:1::/64 tunnel-ip1
  201:0:1::/64 tunnel-ip2
  201:0:1::/64 tunnel-ip3
  201:0:1::/64 tunnel-ip4
 !
!
router ospf 10
 router-id 1.1.1.1
 area 0
  !  interface Bundle-Ether28
  interface Loopback0
  !
  interface HundredGigE0/0/0/26
  !
 !
!
!  interface HundredGigE0/0/0/27
!  interface HundredGigE0/0/0/27.27
router bgp 200
 bgp router-id 1.1.1.1
 address-family ipv4 unicast
  maximum-paths ibgp 64
 !
 ! redistribute connected
 ! neighbor 26.0.1.2
 !  remote-as 200
 !  address-family ipv4 unicast
 !  multipath
 !  route-policy pass in
 !  route-policy pass out
 !  next-hop-self
 neighbor 27.0.1.2
  remote-as 200
  address-family ipv4 unicast
   multipath
   route-policy pass in
   route-policy pass out
   next-hop-self
  !
 !
 neighbor 28.0.1.2
  remote-as 200
  address-family ipv4 unicast
   multipath
   route-policy pass in
   route-policy pass out
   next-hop-self
  !
 !
 neighbor 29.0.1.2
  remote-as 200
  address-family ipv4 unicast
   multipath
   route-policy pass in
   route-policy pass out
   next-hop-self
  !
```

```
 !
!
vrf-policy
 vrf default address-family ipv4 policy type pbr input magic-decap
!
l2vpn
 bridge group bg
  bridge-domain bd
 !   interface HundredGigE0/0/0/29
 !    static-mac-address 0000.1122.2929
 !   routed interface BVI29
 bridge group bg1
  bridge-domain bd1
   interface HundredGigE0/0/0/23
    static-mac-address 0000.1122.2323
   !
   routed interface BVI23
   !
  !
 !
!
mpls static
 interface HundredGigE0/0/0/24
 lsp gre
  in-label 35001 allocate per-prefix 202.0.1.2/32
  forward
   path 1 nexthop tunnel-ip1 out-label 35002
   path 2 nexthop tunnel-ip2 out-label 35002
  !
 !
!
ssh server vrf default
hw-module profile gue udp-dest-port ipv4 1001 ipv6 1002 mpls 1003
end
```

Verification

Run the **show ofa objects sys location 0/0/CPU0 | inc gue** command in the XR Config mode to verify that the unique GUE port numbers have been configured to decapsulate IPv4, IPv6, and MPLS payloads.

```
Router#show ofa objects sys location 0/0/CPU0 | inc gue
uint32_t gue_ipv4_port => 1001
uint32_t gue_ipv6_port => 1002
uint32_t gue_mpls_port => 1003
```