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
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 1

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.

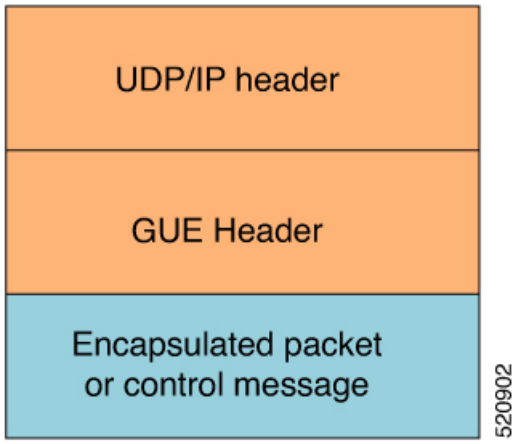
Generic UDP Encapsulation (GUE) is a UDP-based network encapsulation protocol that encapsulates IPv4 and IPv6 packets. GUE provides native UDP encapsulation and defines an additional header, which helps to

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.

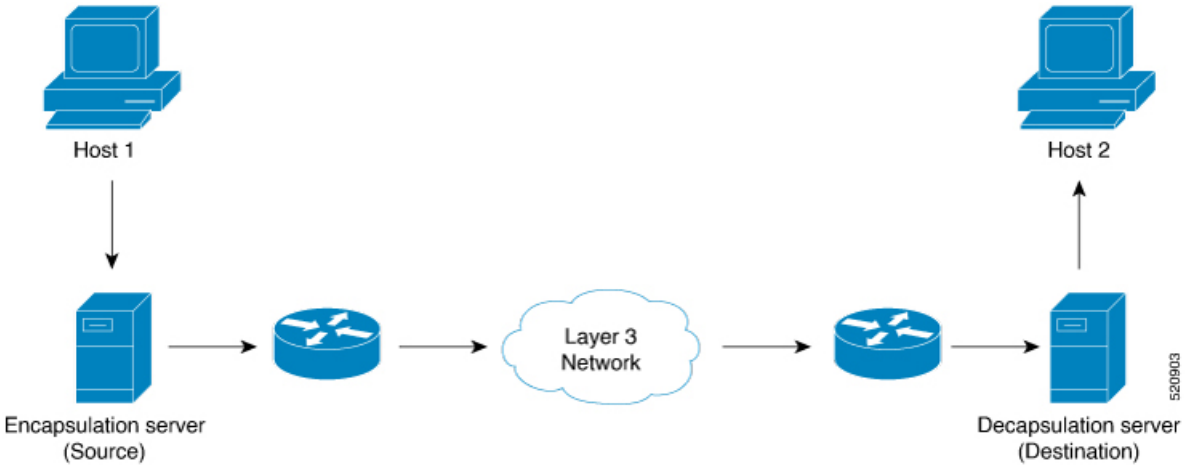
In GUE, the payload is encapsulated in an IP packet that can be IPv4 or IPv6 Carrier. The UDP header is added to provide extra hashing parameters, and optional payload demultiplexing. At the decapsulation node, the Carrier IP and UDP headers are removed, and the packet is forwarded based on the inner 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 6615.
- 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.

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
```