Understand Generic UDP Encapsulation

UDP encapsulation is a technique of adding network headers to the 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.

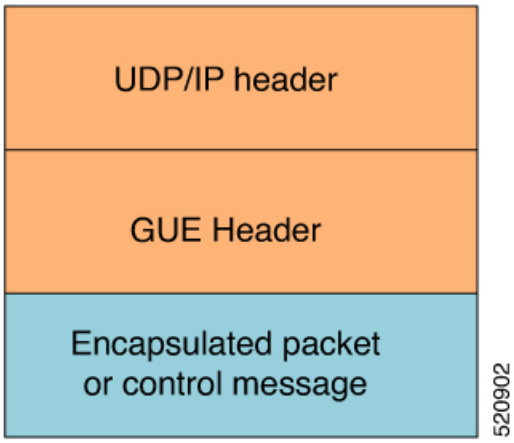
Traditional mechanisms like Generic Routing Encapsulation (GRE) can only handle the outer Source IP address and parts of the destination address and may not provide sufficient load balance entropy.

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, that 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.

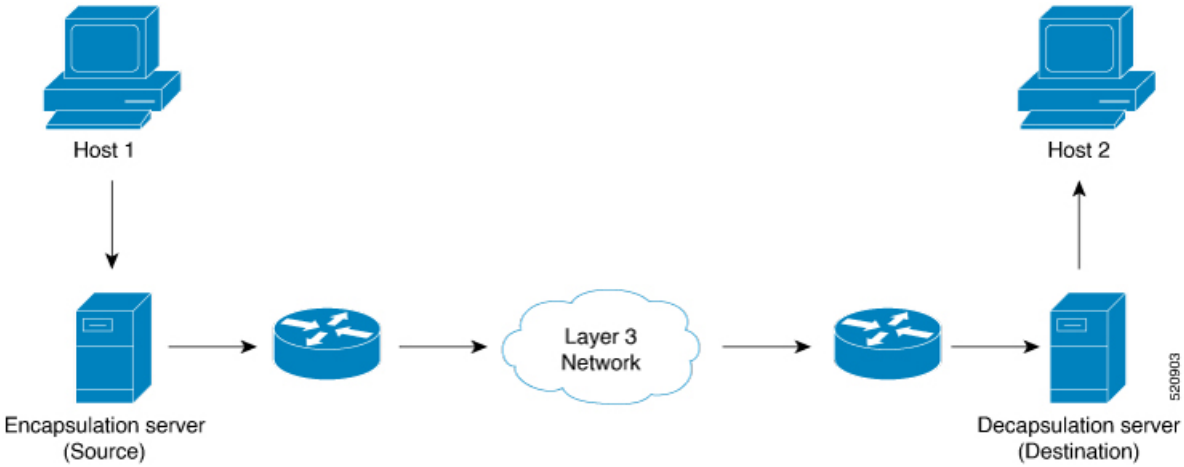
In GUE, the payload is encapsulated in an IP packet that can be IPv4 or IPv6 Carrier. The UDP header is added to provide extra hashing parameters, and optional payload demultiplexing. At the decapsulation node, the Carrier IP and UDP headers are removed, and the packet is forwarded based on the inner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

A GUE packet has the general format:

Figure 1: GUE Packet Format



For example, if the data stream is sent from Host 1 to Host 2. The server acts as a GUE encapsulator that is sending 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 2 bits of the version number in the IP header.

Benefits of using GUE

- Allows direct encapsulation of payloads like 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. Destination also could be terminated based on a subnet providing additional bits for entropy.
- Avoids special handling for tranist 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, on page 2
- Configure GUE, on page 3
- Flexible Assignment of UDP Port Numbers for Decapsulation, on page 5

Restrictions

- Supports Generic UDP Decapsulation for variant 1 only.
- 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.
- Destination Port is mandatory to perform decapsulation.
- 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

Use the following configuration work flow to configure GUE, which is required to decode an incoming GUE packet on router:

1. 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.
2. Configure a policy map: Define a policy map and associate the traffic class with the traffic policy.
3. Apply the policy per VRF basis, and apply this policy on all the interfaces that are part of the VRF.

Configuration Example

1. Configure a traffic class:

```
Router# configure
Router(config)# class-map type traffic match-all gre-1
Router(config-cmap)# match destination-address ipv4 225.100.20.0 255.255.255.0
Router(config-cmap)# match protocol gre
Router(config-cmap)# end-class-map
Router(config)# commit

Router(config)# class-map type traffic match-all udp-v4
Router(config-cmap)# match destination-address ipv4 220.100.20.0 255.255.255.0
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.0
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 220.100.20.0 255.255.255.0
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.0
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 220.100.20.0 255.255.255.0
Router(config-cmap)# match source-address ipv4 210.100.20.0 255.255.255.0
Router(config-cmap)# match protocol udp
Router(config-cmap)# match destination-port 6615
Router(config-cmap)# end-class-map
Router(config)# commit
```

2. 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 gre-1
Router(config-pmap-c)# decapsulate gre
Router(config-pmap-c)# exit

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
```

3. Apply the policy per VRF basis:

```
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
```

Configure Generic UDP Decapsulation for Load Balancing

On transit routers, the outer IP for hashing is used to encode the entropy parameters. But at the terminating or decapsulating router, the payload is used for hashing. However, you can use the outer IP at the decapsulating router as well, as payloads may have limited entropy. To enable the outer IP based hashing on the decapsulation router, use this command:

```
Router(config)# hw-module profile load-balance algorithm ip-tunnel
Router(config)# commit
```

Note Unlike other **hw-module** commands, the **hw-module profile load-balance algorithm ip-tunnel** command requires a reload of the system.

Flexible Assignment of UDP Port Numbers for Decapsulation

Table 1: 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.

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.

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
```