Controlling the TTL Value of Inner Payload Header

Cisco 8000 Routers allow you to control the TTL value of inner payload header of IP-in-IP tunnel packets before it gets forwarded to the next-hop router. This feature enables a router to forward custom formed IP-in-IP stacked packets even if the inner packet TTL is 1. Therefore, this feature enables you to measure the link-state and path reachability from end to end in a network.

Note After you enable or disable the decrement of the TTL value of the inner payload header of a packet, you do not need to reload the line card.

Configuration

To disable the decrement of the TTL value of inner payload header of an IP-in-IP packet, use the following steps:

1. Enter the global configuration mode.
2. Disable the decrement of TTL value of inner payload header of an IP-in-IP packet.

Configuration Example

```
/* Enter the Global Configuration mode. */
Router# configure

/* Disable the decrement of TTL value of inner payload header of an IP-in-IP packet. */
Router(config)# hw-module profile cef ttl tunnel-ip decrement disable
Router(config)# commit
```

Associated Commands

- hw-module profile cef ttl tunnel-ip decrement disable

- IP-in-IP Decapsulation, on page 2
- ECMP Hashing Support for Load Balancing, on page 10

IP-in-IP Decapsulation

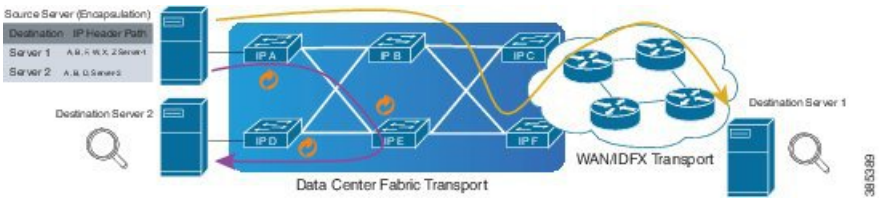
IP-in-IP encapsulation involves the insertion of an outer IP header over the existing IP header. The source and destination address in the outer IP header point to the endpoints of the IP-in-IP tunnel. The stack of IP headers is used to direct the packet over a predetermined path to the destination, provided the network administrator knows the loopback addresses of the routers transporting the packet. This tunneling mechanism can be used for determining availability and latency for most network architectures. It is to be noted that the entire path from source to the destination does not have to be included in the headers, but a segment of the network can be chosen for directing the packets.

In IP-in-IP encapsulation and decapsulation has two types of packets. The original IP packets that are encapsulated are called Inner packets and the IP header stack added while encasulation are called the Outer packets.

Note The router only supports decapsulation and no encapsulation. Encapsulation is done by remote routers.

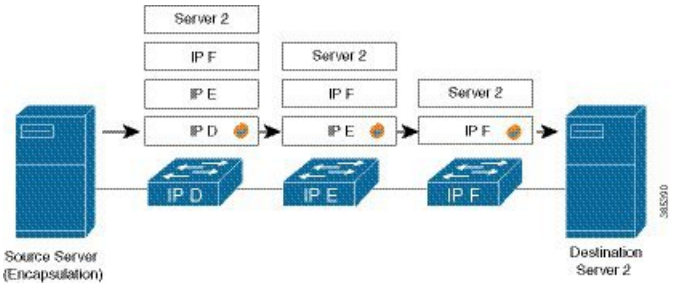
The following topology describes a use case where IP-in-IP encapsulation and decapsulation are used for different segments of the network from source to destination. The IP-in-IP tunnel consists of multiple routers that are used to decapsulate and direct the packet through the data center fabric network.

Figure 1: IP-in-IP Decapsulation Through a Data Center Network



The following illustration shows how the stacked IPv4 headers are decapsulated as they traverse through the decapsulating routers.

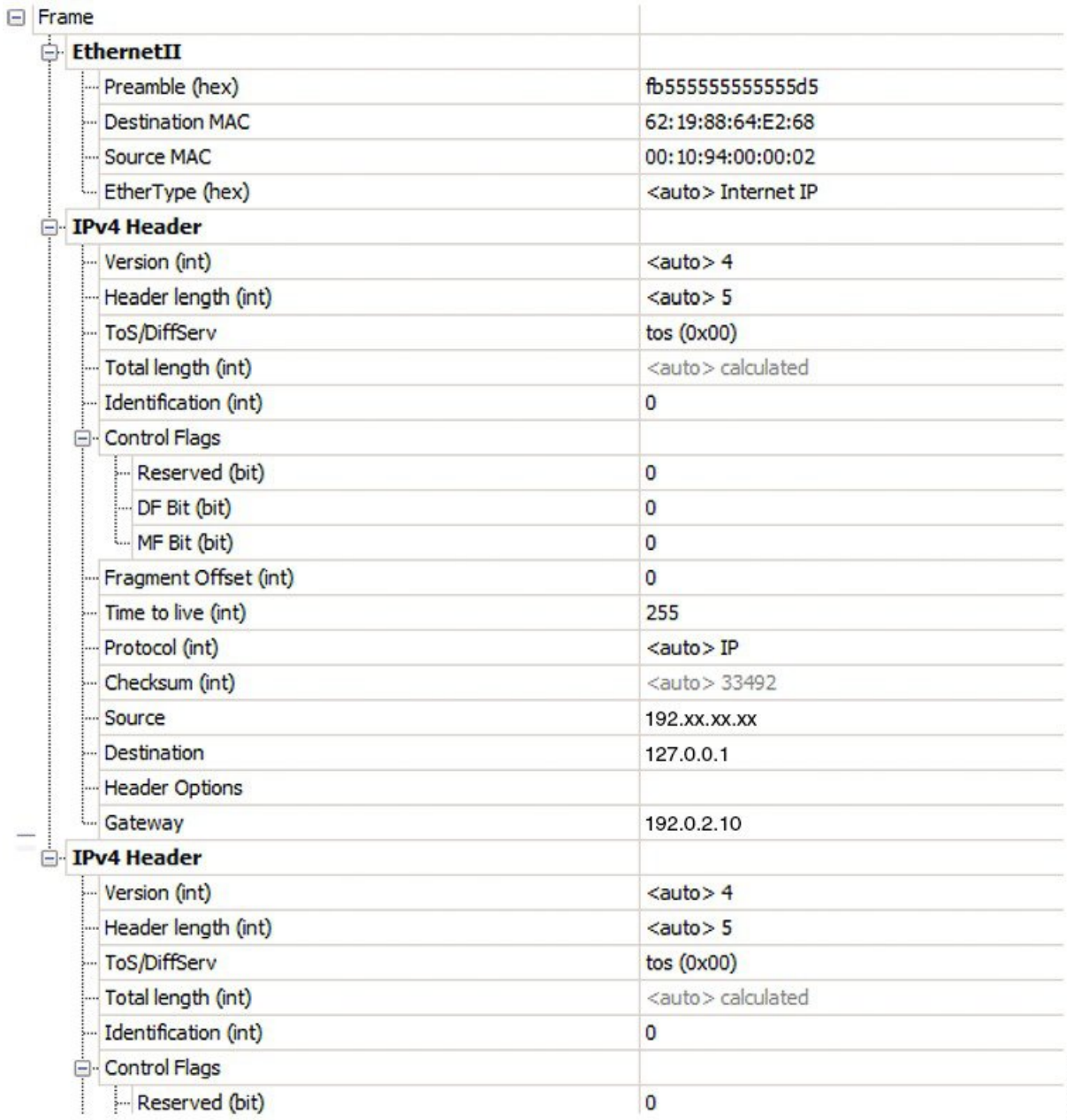
Figure 2: IP Header Decapsulation



Stacked IP Header in an Encapsulated Packet

The encapsulated packet has an outer IPv4 header that is stacked over the original IPv4 header, as shown in the following illustration.

Figure 3: Encapsulated Packet



Configuration

You can use the following sample configuration in the routers to decapsulate the packet as it traverses the IP-in-IP tunnel:

```
Router(config)# interface loopback 0
Router(config-if)# ipv4 address 127.0.0.1/32
Router(config-if)# no shutdown
Router(config-if)# interface tunnel-ip 10
```

```
Router(config-if)# ipv4 unnumbered loopback 1
Router(config-if)# tunnel mode ipv4 decap
Router(config-if)# tunnel source loopback 0
```

- **tunnel-ip**: configures an IP-in-IP tunnel interface.
- **ipv4 unnumbered loopback address**: enables ipv4 packet processing without an explicit address, except for loopback address.
- **tunnel mode ipv4 decap**: enables IP-in-IP decapsulation.
- **tunnel source**: indicates the source address for the IP-in-IP decap tunnel with respect to the router interface.

Note You can configure the tunnel destination only if you want to decapsulate packets from a particular destination. If no tunnel destination is configured, then all the ip-in-ip ingress packets on the configured interface are decapsulated.

Running Configuration

```
Router# show running-config interface tunnel-ip 10
...
interface tunnel-ip 10
ipv4 unnumbered loopback 1
tunnel mode ipv4 decap
```

Extended ACL to Match the Outer Header for IP-in-IP Decapsulation

Starting with Cisco IOS XR Software Release 7.0.14, extended ACL has to match on the outer header for IP-in-IP Decapsulation. Extended ACL support reduces mirrored traffic throughput. This match is based only on the IPv4 protocol, and extended ACL is applied to the received outermost IP header, even if the outer header is locally terminated.

Sample configuration:

```
Router#show running-config interface bundle-Ether 50.5
Tue May 26 12:11:49.017 UTC
interface Bundle-Ether50.5
ipv4 address 101.1.5.1 255.255.255.0
encapsulation dot1q 5
ipv4 access-group ExtACL_IPinIP ingress
ipv4 access-group any_dscpegg egress
!


Router#show access-lists ipv4 ExtACL_IPinIP hardware ingress location$
Tue May 26 12:11:55.940 UTC
ipv4 access-list ExtACL_IPinIP
10 permit ipv4 192.168.0.0 0.0.255.255 any ttl gt 150
11 deny ipv4 172.16.0.0 0.0.255.255 any fragments
12 permit ipv4 any any
```

Decapsulation Using Tunnel Source Direct

Table 1: Feature History Table

Feature Name	Release Information	Feature Description
Decapsulating Using Tunnel Source Direct	Release 7.5.3	Tunnel source direct allows you to decapsulate the tunnels on any L3 interface on the router. You can use the **tunnel source direct** configuration command to choose the specific IP Equal-Cost Multipath (ECMP) links for troubleshooting, when there are multiple IP links between two devices.

To debug faults in various large networks, you may have to capture and analyze the network traffic at a packet level. In datacenter networks, administrators face problems with the volume of traffic and diversity of faults. To troubleshoot faults in a timely manner, DCN administrators must identify affected packets inside large volumes of traffic. They must track them across multiple network components, analyze traffic traces for fault patterns, and test or confirm potential causes.

In some networks, IP-in-IP decapsulation is currently used in network management, to verify ECMP availability and to measure the latency of each path within a datacenter.

The Network Management System (NMS) sends IP-in-IP (IPv4 or IPv6) packets with a stack (multiple) of predefined IPv4 or IPv6 headers (device IP addresses). The destination device at each hop removes the outside header, performs a lookup on the next header, and forwards the packets if a route exists.

Using the **tunnel source direct** command, you can choose the specific IP Equal-Cost Multipath (ECMP) links for troubleshooting, when there are multiple IP links between two devices.

Tip You can programmatically configure and manage the Ethernet interfaces using `openconfig-ethernet-if.yang` and `openconfig-interfaces.yang` OpenConfig data models. To get started with using data models, see the *Programmability Configuration Guide for Cisco 8000 Series Routers*.

Guidelines and Limitations

The following guidelines are applicable to this feature.

- The **tunnel source direct** command is supported only with the tunnel mode as **decap** (when an administrator uses the IP-in-IP decapsulation).
- The source-direct tunnel is always operationally `up` unless it is administratively shut down. The directly connected interfaces are identified using the **show ip route direct** command.
- The **tunnel source direct** command is supported only in IP-in-IP tunneling `decap` mode.
- All Layer 3 interfaces that are configured on the device are supported.

- Platform can accept and program only certain number of IP addresses. The number of IP addresses depends on the make of the platform linecard (LC). Each LC can have different number of Network Processor (NP) slices and interfaces.
- Any inline modification of the **tunnel source direct** command like **tunnel source** *interface | IP address* is not supported. You must delete the tunnel and recreate it.
- Only one source-direct tunnel per address-family is supported for configuration.
- Regular decapsulation tunnels which have specific source address, are supported. However, the tunnel's specific source address must not be part of any interface.

The following functionalities are not supported for the **tunnel source direct** option.

- GRE tunneling mode.
- VRF (only default VRF is supported).
- ACL and QoS on the tunnels.
- Tunnel encapsulation.
- Tunnel NetIO DLL: Decapsulation is not supported if the packet is punted to slow path.

Configuration

The **tunnel source direct** configures IP-in-IP tunnel decapsulation on any directly connected IP addresses. This option is now supported only when the IP-in-IP decapsulation is used to source route the packets through the network.

This example shows how to configure IP-in-IP tunnel decapsulation on directly connected IP addresses:

```
Router# configure terminal
Router(config)#interface Tunnel4
  Router(config)#tunnel mode ipv4 decap
  Router(config)#tunnel source direct
  Router(config)#no shutdown
```

This example shows how to configure IP-in-IP tunnel decapsulation on IPv6 enabled networks:

```
Router# configure terminal
Router(config)#interface Tunnel6
  Router(config)#tunnel mode ipv6 decap
  Router(config)#tunnel source direct
  Router(config)#no shutdown
```

Verifying the Configuration

The following example shows how to verify IP-in-IP tunnel decapsulation with **tunnel source direct** option:

```
Router#show running-config interface tunnel 1
interface Tunnel1
  tunnel mode ipv6ipv6 decapsulate-any
  tunnel source direct
  no shutdown

Router#show interface tunnel 1
Tunnel1 is up    Admin State: up
MTU 1460 bytes, BW 9 Kbit
Tunnel protocol/transport IPv6/DECAPANY/IPv6
```

```
Tunnel source - direct
Tx    0 packets output, 0 bytes    Rx    0 packets input, 0 bytes
```

Configure Tunnel Destination with an Object Group

Table 2: Feature History Table

Feature Name	Release Information	Description
Configure Tunnel Destination with an Object Group	Release 7.5.4	You can now assign an object group as the destination for an IP-in-IP decapsulation tunnel. With this functionality, you could configure an IPv4 or IPv6 object group consisting of multiple IPv4 or IPv6 addresses as the destination for the tunnel instead of a single IPv4 or IPv6 address. Using an object group instead of a singular IP address. This helps reduce the configuration complexity in the router by replacing the multiple tunnels with one destination with a single decapsulation tunnel that supports a diverse range of destinations The feature introduces these changes: • **CLI**: New **tunnel destination** command. • **YANG Data Model**: New **object-group** option supported in Cisco-IOS-XR-um-if-tunnel-cfg.yang Cisco native model (see GitHub).

In IP-in-IP Decapsulation, the router accepts a packet on a tunneled interface only when the tunnel IP address matches the source IP address of the incoming packets. With this implementation, the user needs to configure separate interface tunnels for each IP address that the router needs to receive the traffic packets. This limitation often leads to configuration overload on the router.

You can eliminate the configuration overload on the router by assigning an object group as the tunnel destination for IPv4 and IPv6 traffic types. That is, the router matches the source IP address of the incoming packet against the object group available as the tunnel destination. The decapsulation tunnel accepts the incoming traffic packets when there's a match between the packet source and the object group. Otherwise, the router drops the packets.

Restrictions

The following restrictions are applicable to the tunnel destination with an object group feature:

- GRE tunnels don't support configuring object groups as the tunnel destination.
- The router supports configuring tunnel destination with an object group only when the tunnel source is tunnel source direct.
- You can configure the object group as tunnel destination only on default VRF.
- Configuring object groups as the tunnel destination isn't applicable to tunnel encapsulation.
- Subinterfaces don't support configuring object groups as the tunnel destination.
- Configuring object groups as the tunnel destination feature is mutually exclusive with ACL and QoS features.
- The tunnel destination feature supports only IPv4 and IPv6 object groups.
- The router does not support changing tunnel configuration after its creation. Configure the tunnel source direct and tunnel destination with an object group while creating the tunnel only.

Prerequisites

- Define an object group including the network elements for the tunnel destination.
- Enable the tunnel source direct feature. For more information, see decapsulation using tunnel source direct.

Configuration Example

This section provides an example for configuring the tunnel destination with an object group:

Configuration

IPv4:

```
Router# configure
/* Configure the IPv4 object group */
Router(config)# object-group network ipv4 Test_IPv4
Router(config-object-group-ipv4)# 192.0.2.0/24
Router(config-object-group-ipv4)# 198.51.100.0/24
Router(config-object-group-ipv4)# 203.0.113.0/24
Router(config-object-group-ipv4)# commit
Router(config-object-group-ipv4)# exit

/* Enters the tunnel configuration mode */
Router(config)# interface tunnel TestIPv4


/* Configures the tunnel mode */
Router(config-if)# tunnel mode ipv4 decap

/* Configures the tunnel to accept all packets with destination address matching the IP
addresses on the router */
Router(config-if)# tunnel source direct

/* Configures the tunnel to accept all packets with destination address that are in the
specified object group */
Router(config-if)# tunnel destination object-group ipv4 Test_IPv4
```

```
Router(config-if)# no shutdown
Router(config-if)# commit
Router(config-if)# exit
```

IPv6:

```
Router# configure
/* Configure the IPv6 object group */
Router(config)# object-group network ipv6 Test_IPv6
Router(config-object-group-ipv6)# 2001:DB8::/32
Router(config-object-group-ipv6)# 2001:DB8::/48
Router(config-object-group-ipv6)# commit
Router(config-object-group-ipv6)# exit

/* Enters the tunnel configuration mode */
Router(config)# interface tunnel TestIPv6


/* Configures the tunnel mode */
Router(config-if)# tunnel mode ipv6 decap

/* Configures the tunnel to accept all packets with destination address matching the IP
addresses on the router */
Router(config-if)# tunnel source direct

/* Configures the tunnel to accept all packets with destination address that are in the
specified object group */
Router(config-if)# tunnel destination object-group ipv6 Test_IPv6

Router(config-if)# no shutdown
Router(config-if)# commit
Router(config-if)# exit
```

Running Configuration

```
Router# show running config object-group
object-group network ipv4 Test_IPv4
192.0.2.0/24
198.51.100.0/24
203.0.113.0/24
!
object-group network ipv6 Test_IPv6
2001:DB8::/32
2001:DB8::/48
!

Router# show interface tunnel TestIPv4
interface TunnelTestIPv4
  tunnel mode ipv4 decap
   tunnel source direct
  tunnel destination object-group ipv4 Test_IPv4
  no shutdown
!

Router# show interface tunnel TestIPv6
interface TunnelTestIPv6
 tunnel mode ipv6 decap
   tunnel source direct
  tunnel destination object-group ipv6 Test_IPv6
 no shutdown
!
```

Verification

```
Router# show tunnel ip ea database

----- node0_0_CPU0 -----
tunnel ifhandle 0x80022cc
tunnel source 161.115.1.2
tunnel destination address group Test_IPv4
tunnel transport vrf table id 0xe0000000
tunnel mode gre ipv4, encap
tunnel bandwidth 100 kbps
tunnel platform id 0x0
tunnel flags 0x40003400
IntfStateUp
BcStateUp
Ipv4Caps
Encap
tunnel mtu 1500
tunnel tos 0
tunnel ttl 255
tunnel adjacency flags 0x1
tunnel o/p interface handle 0x0
tunnel key 0x0, entropy length 0 (mask 0xffffffff)
tunnel QT next 0x0
tunnel platform data (nil)
Platform:
Handle: (nil)
Decap ID: 0
Decap RIF: 0
Decap Recycle Encap ID: 0x00000000
Encap RIF: 0
Encap Recycle Encap ID: 0x00000000
Encap IPv4 Encap ID: 0x4001381b
Encap IPv6 Encap ID: 0x00000000
Encap MPLS Encap ID: 0x00000000
DecFEC DecRcyLIF DecStatsId EncRcyLIF
```

ECMP Hashing Support for Load Balancing

The system inherently supports the n-tuple hash algorithm. The first inner header in the n-tuple hashing includes the source port and the destination port of UDP / TCP protocol headers.

The load balancing performs these functions:

- Incoming data traffic is distributed over multiple equal-cost connections.
- Incoming data traffic is distributed over multiple equal-cost connections member links within a bundle interface.
- Layer 2 bundle and Layer 3 (network layer) load-balancing decisions are taken on IPv4, and IPv6. If it is an IPv4 or an IPv6 payload, then an n-tuple hashing is done.
- An n-tuple hash algorithm provides more granular load balancing and used for load balancing over multiple equal-cost Layer 3 (network layer) paths. The Layer 3 (network layer) path is on a physical interface or on a bundle interface.
- The n-tuple load-balance hash calculation contains:
    - Source IP address
    - Destination IP address

- IP Protocol type
- Router ID
- Source port
- Destination port
- Input interface
- Flow-label (for IPv6 only)

User-Defined Fields for ECMP Hashing

Table 3: Feature History Table

Feature Name	Release Information	Description
User-Defined Fields for ECMP Hashing	Release 7.5.5	We ensure that in cases where multiple paths are used to carry packets from source to destination, each path is utilized for this purpose and no path is over-utilized or congested. This is made possible because we now provide customized ECMP hashing fields that are used for path computation. Previously, the router relied on fixed packet header fields for hashing, which were not user configurable. With additional user-defined bytes considered for hashing, the granularity at which the traffic can be analyzed for ECMP load balancing increases, resulting in better load balancing and path utilization. The feature introduces these changes: **CLI:** • **cef load-balancing fields user-data** • The **show cef exact-route** command is modified with a new **user-data** keyword. • The **show cef ipv4 exact-route** command is modified with a new **user-data** keyword. • The **show cef ipv6 exact-route** command is modified with a new **user-data** keyword. **YANG:** • New Xpath for `Cisco-IOS-XR-8000-fib-platform-cfg.yang` (see Github, YANG Data Models Navigator).

ECMP hashing is used to distribute traffic across multiple equal-cost paths. See ECMP Hashing Support for Load Balancing for the default static hashing algorithm details.

You can now add user-defined packet header fields for ECMP path calculation for ipv4 and ipv6 flows using the **cef load-balancing fields user-data** command. Ensure you specify these user-defined fields based on the type of traffic flow that requires load balancing. You can include the following parameters:

- **Hash header:** The hash header specifies which packet header is being considered for load balancing. You can enable any or all of the available six profiles.
  - IPv4: tcp, udp, non-tcp-udp
  - IPv6: tcp, udp, non-tcp-udp

  If any hash header profile is defined for load balancing, along with the fixed fields considered for hashing, additional bytes in the payload are also used for path computation.
- **Hashing offset:** The hashing offset specifies the byte location from the end of the configured header.
- **Hash size:** The hash size specifies the number of bytes that is considered from the start of the hash offset by the ECMP hashing algorithm. Range is 1 to 4 bytes.
- **Location:** This specifies the location of the ingress line card that receives the incoming traffic. The user-defined hashing configuration is applied on the specified line card.

The addition of the user-defined packer header fields increases the granularity at which the traffic is analyzed for ECMP load balancing. When multiple paths with equal cost are available for routing a specific type of packet from a source to a destination, this granularity ensures that the intended type of traffic is evenly distributed across these paths. This ensures all available paths are used efficiently and prevents congestion or over-utilization of a single path.

You can also retrieve the exact-route information based on the configured user-data using the **show cef exact-route** command with **user-data** keyword.

Note

- When the user-defined hashing configuration is active, any additional options or optional keywords are disregarded during the parsing of incoming packets for retrieving the user-defined bytes.
- The hashing results based on user-defined hash feature is applicable to BGP/IGP ECMP and LAG hashing.
- The use of the user-defined hashing configuration changes the load balancing behavior of GRE and IPinIP traffic. This includes all traffic that begins with ipv4, ipv6, ipv4+udp, ipv6+udp, ipv4+tcp, and ipv6+tcp, regardless of the payload.

Configure User-Defined Fields for ECMP Hashing

The command **cef load-balancing fields user-data** configures the additional user-defined fields that are to be considered for the hashing algorithm.

This example shows how to configure the additional IPv4 header fields for TCP packets:

```
Router# configure terminal
Router(config)#cef load-balancing fields user-data ipv4 tcp offset 5 size 3 location 0/0/CPU0
Router(config)#commit
```

- offset 5: The payload considered for hashing starts from byte 6 from the end of TCP header.
- size 3: Three bytes of payload are considered.
- location 0/0/CPU0: Specifies the line card on which the configuration is applied.

In the above example, the sixth, seventh, and eighth bytes of the payload are considered additionally for the hashing.

This example shows how to configure the additional IPv6 header fields for UDP packets:

```
Router# configure terminal
Router(config)#cef load-balancing fields user-data ipv6 udp offset 0 size 2 location 0/0/CPU0
Router(config)#commit
```

- offset 0: The payload considered for hashing starts from the end of UDP header.
- size 2: Two bytes of payload are considered.
- location 0/0/CPU0: Specifies the line card on which the configuration is applied.

In the above example, the first two bytes of payload of a UDP packet are considered additionally for the hashing.

Running Configuration

The following example shows the running configuration:

```
Router#show running-config | include cef
Fri Jul 28 12:02:01.002 UTC
cef load-balancing fields user-data ipv4 tcp offset 5 size 3 location 0/0/CPU0
cef load-balancing fields user-data ipv6 udp offset 0 size 2 location 0/0/CPU0
Router#
```

Verification

The following example shows the difference in load balancing before and after applying user-defined hashing, for a flow with data that exhibits good hashing behavior.

Before applying user-defined hashing

```
Router#show interfaces accounting | i IPV6_U
  Protocol              Pkts In         Chars In     Pkts Out        Chars Out
  IPV6_UNICAST               1               72            0                0
  IPV6_UNICAST               1               72            0                0
  IPV6_UNICAST               1               72            0                0
  IPV6_UNICAST               1               72            0                0
  IPV6_UNICAST               2              144            0                0
  IPV6_UNICAST               1               72            0                0
  IPV6_UNICAST               0                0      3979416       1981749168
  IPV6_UNICAST         4191438       2087336124            0                0
  IPV6_UNICAST               1               72            0                0
  IPV6_UNICAST               1               72            0                0
  IPV6_UNICAST               1               72            0                0
Router#
```

After applying user-defined hashing

```
Router#show interfaces accounting | i IPV6_U
  Protocol              Pkts In         Chars In     Pkts Out        Chars Out
  IPV6_UNICAST               0                0        39119         19481262
  IPV6_UNICAST               0                0        39801         19820898
  IPV6_UNICAST               0                0        40483         20160534
```

```
  IPV6_UNICAST                0                0        40524          20180952
  IPV6_UNICAST                0                0        40573          20205354
  IPV6_UNICAST                0                0        40614          20225772
  IPV6_UNICAST                0                0        39368          19605264
  IPV6_UNICAST                0                0        40734          20285532
  IPV6_UNICAST                0                0        40777          20306946
  IPV6_UNICAST                0                0        40171          20005158
  IPV6_UNICAST                0                0        40858          20347284
  IPV6_UNICAST                0                0        40269          20053962
  IPV6_UNICAST                0                0        41603          20718294
  IPV6_UNICAST                0                0        40363          20100774
  IPV6_UNICAST                0                0        40407          20122686
  IPV6_UNICAST                0                0        41098          20466804
  IPV6_UNICAST           850393        423495714            0                0
```

To view the exact route information allocated to the packets, use **show cef exact-route** command with **user-data** keyword.

The packet contains value 0x2 in the packet position for the ipv6 packet, for which the user-defined configuration has been added for a non-tcp-udp ipv6 flow.

```
Router#show cef ipv6 exact-route 100::10 60::1 flow-label 0 protocol 59 source-port 0
destination-port 0 user-data 0x2 ingress-interface HundredGigE0/0/0/2 location 0/0/cpu0
Unsupported protocol value 59
60::/16, version 1293, internal 0x1000001 0x20 (ptr 0x8b78ef00) [1], 0x400 (0x8e9cfc48),
0x0 (0x0)
 Updated Aug 14 07:50:20.022
 local adjacency to Bundle-Ether3.30

 Prefix Len 16, traffic index 0, precedence n/a, priority 2
   via Bundle-Ether3.30
   via fe80::72b3:17ff:feae:d703/128, Bundle-Ether3.30, 7 dependencies, weight 0, class 0
[flags 0x0]
    path-idx 7 NHID 0x0 [0x8db8bed8 0x0]
    next hop fe80::72b3:17ff:feae:d703/128
    local adjacency
```