# MTU Behavior on Cisco IOS XR and Cisco IOS Routers

## Contents


**Introduction**

**Background Information**

- Comparison of Cisco IOS and Cisco IOS XR Software
- Routed L3 Interfaces
  - Default MTU
  - Non-Default MTU
- Routed L3 Sub-Interfaces
- L2VPN L2 Interface
  - EVC (ASR9000)
  - Non EVC (XR 12000 and CRS)

**Automatic Ethernet Interface Driver MTU and MRU Configuration**

- Convert the Configuration When You Upgrade from a Release Earlier than Release 5.1.1 to Release 5.1.1 or Later


## Introduction

This document describes maximum transmission unit (MTU) behaviors on Cisco IOS® XR routers and compares those behaviors to Cisco IOS routers. It also discusses MTUs on routed Layer 3 (L3) interfaces and Layer 2 VPN (L2VPN) L2 interfaces that use both the Ethernet Virtual Connection (EVC) and non-EVC models. This document also describes important changes to how Ethernet interface driver MTU and maximum receive unit (MRU) are automatically configured in Release 5.1.1 and later.

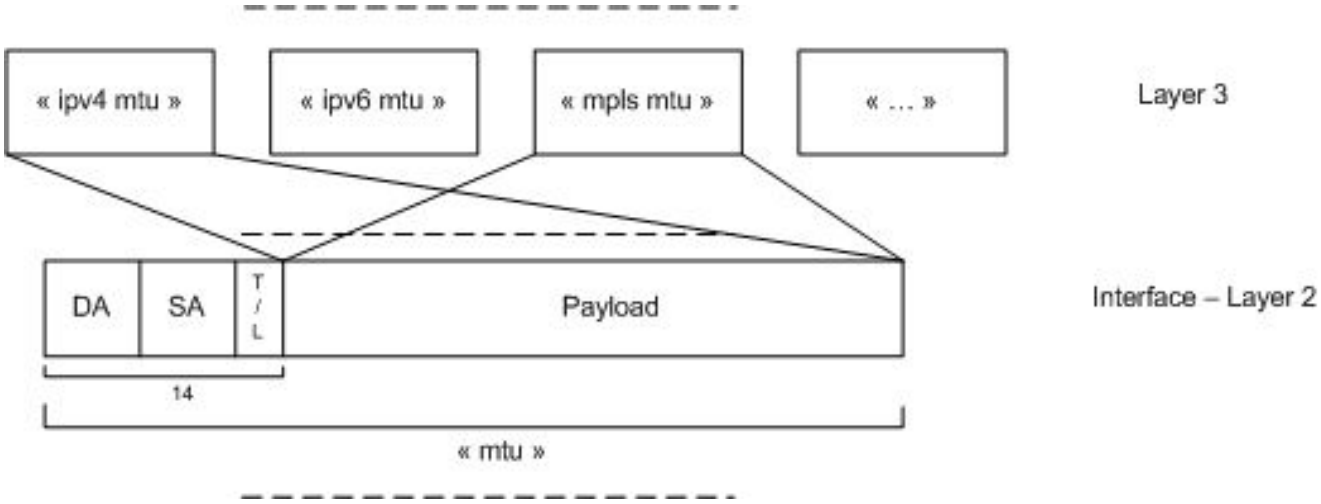
## Background Information

In computer networking, the MTU of a communications protocol of a layer defines the size, in bytes, of the largest protocol data unit that the layer is allowed to transmit over one interface. One MTU parameter is associated with each interface, layer, and protocol.

MTU characteristics in Cisco IOS XR software are:

- MTU **config** and **show** commands, at L2 and L3, include the header size of their corresponding layer. For instance, the **mtu** command that configures the L2 MTU includes 14 bytes for an Ethernet interface (without dot1q), or 4 bytes for Point-to-Point Protocol (PPP) or high-level data link control (HDLC). The **ipv4 mtu** command includes 20 bytes of the IPv4 header.

- The MTU of a higher layer must fit within the payload of the lower layer. For example, if the interface MTU of a non-dot1q Ethernet interface is the default of 1514 bytes, then higher layer protocols such as Multiprotocol Label Switching (MPLS) can have a maximum MTU of 1500 bytes on that interface. This means you can fit only a 1500 byte MPLS frame (including labels) inside the Ethernet frame. You cannot configure a 1508 byte MPLS MTU on that interface if you want to allow two MPLS tags on top of a 1500 byte IPv4 packet. In order to transmit a 1508 byte MPLS frame on an Ethernet interface, the interface MTU must be increased to 1522, or higher value, in order to ensure that the L2 interface payload is large enough to carry the MPLS frame.




- In the classical Cisco IOS software (not the Cisco IOS XR software), the interface **mtu** command configures the L2 payload size, but does not include the L2 header. This is different from the Cisco IOS XR software which includes both the L2 and L3 overhead in the **interface mtu** command. The L3 MTU commands, as in the case of the **ipv4 mtu** command, configure the maximum packet size of that protocol which includes the L3 header. This is similar to the case of Cisco IOS XR software.

- The default interface MTU in the Cisco IOS XR software must allow the transport of a 1500 byte L3 packet. Therefore, the default MTU is 1514 bytes for a main Ethernet interface and 1504 bytes for a serial interface.

The remainder of this document illustrates MTU characteristics, compares Cisco IOS and Cisco IOS XR software behavior, and gives examples for these types of interfaces:

- Routed L3 interfaces
- Routed L3 sub-interfaces
- L2VPN L2 interfaces

## Comparison of Cisco IOS and Cisco IOS XR Software

This section compares Cisco IOS and Cisco IOS XR software behavior with reference to MTU characteristics.

In Cisco IOS software, the **mtu** command and the corresponding **show** commands do not include the L2 header. Use the **mtu** command in order to configure the L2 payload to the maximum size for the L3 packets, including the L3 header.

This is different from Cisco IOS XR software, where the **mtu** command includes the L2 header (14 bytes for Ethernet or 4 bytes for PPP/HDLC).

If a Cisco IOS router is configured with **mtu** *x* and is connected to a Cisco IOS XR router, then the corresponding interface on the Cisco IOS XR router should be configured with **mtu** *x***+14** for Ethernet interfaces, or **mtu** *x***+4** for serial interfaces.

Cisco IOS and Cisco IOS XR software have the same meaning for the **ipv4 mtu**, **ipv6 mtu** and **mpls mtu** commands; they must be configured with the same values.

As a result, this is the configuration in Cisco IOS software on an Ethernet interface:

```
mtu 9012
ipv4 mtu 9000
ipv6 mtu 9000
```

The corresponding configuration on the Cisco IOS XR software neighbor is:

```
mtu 9026
ipv4 mtu 9000
ipv6 mtu 9000
```

## Routed L3 Interfaces

The MTU values must be the same on all devices connected to an L2 network. Otherwise, these symptoms might be reported:

- Intermediate System-to-Intermediate System (IS-IS) adjacencies do not come up. By default, IS-IS uses hello-padding; therefore, hellos might be characterized as giants and might be dropped when one router has an MTU value that is lower than the values at the other routers.
- Open Shortest Path First (OSPF) adjacencies get stuck in Exstart or Exchange state, because large database descriptor (DBD) packets might be characterized as giants and might be dropped. When the packets are received on a router with a lower MTU value, the databases are not synchronized.
- Data traffic is characterized as giants and is dropped when it is received on a device with an MTU value that is lower than the one at the transmitting device.
- There is low throughput when large packets get dropped. In case of path MTU discovery, the TCP session can recover when large packets are dropped, but this affects the throughput.

### Default MTU

This section analyzes the default MTU of a routed interface when the **mtu** command is not configured:

<#root>

```
RP/0/RP0/CPU0:motorhead#
```

**sh run int gigabitEthernet 0/1/0/3**

```
interface GigabitEthernet0/1/0/3
 cdp
 ipv4 address 10.0.1.1 255.255.255.0
 ipv6 address 2001:db8::1/64
!

RP/0/RP0/CPU0:router#
```

**sh int gigabitEthernet 0/1/0/3 | i MTU**

**MTU 1514 bytes**

```
, BW 1000000 Kbit (Max: 1000000 Kbit)
```

```
RP/0/RP0/CPU0:router#
show im database interface gigabitEthernet 0/1/0/3



View: OWN - Owner, L3P - Local 3rd Party, G3P - Global 3rd Party,
      LDP - Local Data Plane, GDP - Global Data Plane, RED - Redundancy

Node 0/1/CPU0 (0x11)

Interface GigabitEthernet0/1/0/3, ifh 0x01180100 (up, 1514)
  Interface flags:          0x000000000010059f (IFCONNECTOR|IFINDEX
                            |SUP_NAMED_SUB|BROADCAST|CONFIG|HW|VIS|DATA
                            |CONTROL)
  Encapsulation:            ether
  Interface type:           IFT_GETHERNET
  Control parent:           None
  Data parent:              None
  Views:                    GDP|LDP|L3P|OWN

  Protocol        Caps (state, mtu)
  --------        -----------------
  None            ether (up,
1514
)
  arp             arp (up,
1500
)
  clns            clns (up,
1500
)
  ipv4            ipv4 (up,
1500
)
  mpls            mpls (up,
1500
)
  ipv6            ipv6_preswitch (up,
1500
)
  ipv6            ipv6 (down,
1500
)
  ether_sock      ether_sock (up,
1500
)

RP/0/RP0/CPU0:router#
```

```
show ipv4 interface gigabitEthernet 0/1/0/3 | i MTU



MTU is 1514 (1500 is available to IP)


RP/0/RP0/CPU0:router#
show ipv6 interface gigabitEthernet 0/1/0/3 | i MTU



MTU is 1514 (1500 is available to IPv6)


RP/0/RP0/CPU0:router#
sh mpls interfaces  gigabitEthernet 0/1/0/3 private location 0/1/CPU0


Interface      IFH         MTU
-------------- ---------- -----
Gi0/1/0/3      0x01180100
1500


RP/0/RP0/CPU0:router#
```

In this example, the default L2 interface MTU is 1514 bytes and includes 14 bytes of Ethernet header. The 14 bytes are accounted by 6 bytes of destination MAC address, 6 bytes of source MAC address, and 2 bytes of type or length. This does not include the preamble, frame delimiter, 4 bytes of frame check sequence (FCS), and inter-frame gap. For a PPP or HDLC frame, 4 bytes of the L2 header are accounted for; so the default interface MTU is 1504 bytes.

The L3 child protocols inherit their MTU from the payload of the parent MTU. When you subtract 14 bytes of an L2 header from an L2 MTU of 1514 bytes, you have an L2 payload of 1500 bytes. This becomes the MTU for L3 protocols. IPv4, IPv6, MPLS, and Connectionless Network Service (CLNS) inherit this1500 byte MTU. As a result, a Cisco IOS XR Ethernet interface, by default, can transport a 1500 byte L3 packet which is the same as the defauIt on a Cisco IOS Ethernet interface.

### Non-Default MTU

This section shows how to configure an **mpls mtu** of 1508 in order to send an IPv4 packet of 1500 bytes with two MPLS tags of 4 bytes each, on top of the packet:

```
<#root>
RP/0/RP0/CPU0:router#
conf


RP/0/RP0/CPU0:router(config)#
int gig 0/1/0/3
```

```
RP/0/RP0/CPU0:router(config-if)#
mpls mtu 1508


RP/0/RP0/CPU0:router(config-if)#
commit


RP/0/RP0/CPU0:Mar 12 00:36:49.807 CET: config[65856]: %MGBL-CONFIG-6-DB_COMMIT : Configuration
   committed by user 'root'. Use 'show configuration commit changes 1000000124' to view the
   changes.
RP/0/RP0/CPU0:router(config-if)#
end


RP/0/RP0/CPU0:Mar 12 00:36:54.188 CET: config[65856]: %MGBL-SYS-5-CONFIG_I : Configured
   from console by root on vty0 (10.55.144.149)

RP/0/RP0/CPU0:router#
sh mpls interfaces  gigabitEthernet 0/1/0/3 private location 0/1/CPU0


Interface      IFH         MTU
-------------- ---------- -----
Gi0/1/0/3      0x01180100
1500


RP/0/RP0/CPU0:router#
show im database interface gigabitEthernet 0/1/0/3



View: OWN - Owner, L3P - Local 3rd Party, G3P - Global 3rd Party,
      LDP - Local Data Plane, GDP - Global Data Plane, RED - Redundancy

Node 0/1/CPU0 (0x11)

Interface GigabitEthernet0/1/0/3, ifh 0x01180100 (up, 1514)
  Interface flags:          0x000000000010059f (IFCONNECTOR|IFINDEX
                            |SUP_NAMED_SUB|BROADCAST|CONFIG|HW|VIS|DATA
                            |CONTROL)
  Encapsulation:            ether
  Interface type:           IFT_GETHERNET
  Control parent:           None
  Data parent:              None
  Views:                    GDP|LDP|L3P|OWN

  Protocol        Caps (state, mtu)
  --------        -----------------
  None            ether (up, 1514)
  arp             arp (up, 1500)
  clns            clns (up, 1500)
  ipv4            ipv4 (up, 1500)
  mpls            mpls (up,
1500
)
  ipv6            ipv6_preswitch (up, 1500)
```

```
  ipv6            ipv6 (down, 1500)
  ether_sock      ether_sock (up, 1500)

RP/0/RP0/CPU0:router#
```

Although the **mpls mtu 1508** command is committed, it is not applied, because MPLS still has an MTU of 1500 bytes in the **show** command. This is because L3 child protocols cannot have an MTU larger than the payload of their parent L2 interface.

In order to allow two labels on top of a 1500 byte IP packet, you must:

- Configure an L2 interface MTU of 1522 bytes, so that all child protocols (including MPLS) inherit an MTU of 1508 bytes (1522 - 14 = 1508).
- Reduce the MTU of the L3 protocols to 1500 bytes, so that only MPLS is allowed to exceed 1500 bytes.

<#root>

```
RP/0/RP0/CPU0:router#
sh run int gig 0/1/0/3


interface GigabitEthernet0/1/0/3
 cdp


mtu 1522




ipv4 mtu 1500


 ipv4 address 10.0.1.1 255.255.255.0


ipv6 mtu 1500


 ipv6 address 2001:db8::1/64
 !
!

RP/0/RP0/CPU0:router#
show im database interface gigabitEthernet 0/1/0/3



View: OWN - Owner, L3P - Local 3rd Party, G3P - Global 3rd Party,
      LDP - Local Data Plane, GDP - Global Data Plane, RED - Redundancy

Node 0/1/CPU0 (0x11)

Interface GigabitEthernet0/1/0/3, ifh 0x01180100 (up,

1522

)
```

```
  Interface flags:          0x000000000010059f (IFCONNECTOR|IFINDEX
                            |SUP_NAMED_SUB|BROADCAST|CONFIG|HW|VIS|DATA
                            |CONTROL)
  Encapsulation:            ether
  Interface type:           IFT_GETHERNET
  Control parent:           None
  Data parent:              None
  Views:                    GDP|LDP|L3P|OWN

  Protocol        Caps (state, mtu)
  --------        -----------------
  None            ether (up,
```

**1522**

```
)
  arp             arp (up, 1508)
  clns            clns (up, 1508)
  ipv4            ipv4 (up,
```

**1500**

```
)
  mpls            mpls (up,
```

**1508**

```
)
  ipv6            ipv6_preswitch (up, 1508)
  ipv6            ipv6 (down,
```

**1500**

```
)
  ether_sock      ether_sock (up, 1508)

RP/0/RP0/CPU0:router#
```

This configuration lets you send IPv4 and IPv6 packets of 1500 bytes and MPLS packets of 1508 bytes (a 1500 byte packet with two tags on the top).

## Routed L3 Sub-Interfaces

These characteristics apply to routed L3 sub-interfaces.

A routed sub-interface MTU inherits the MTU of its parent main interface; add 4 bytes for each VLAN tag configured on the sub-interface. Thus, there are 4 bytes for a dot1q sub-interface and 8 bytes for a IEEE 802.1Q tunneling (QinQ) sub-interface.

As a result, L3 packets of the same size can be forwarded both on the main interface and the sub-interface.

The **mtu** command can be configured under the sub-interface, but it is applied only if it is lower or equal to the MTU that is inherited from the main interface.

This is an example where the MTU of the main interface is 2000 bytes:

<#root>

```
RP/0/RP0/CPU0:router#
```

**sh run int gig 0/1/0/3**

```
interface GigabitEthernet0/1/0/3
 cdp
 mtu 2000
!

RP/0/RP0/CPU0:router#
```

**sh run int gig 0/1/0/3.100**

```
interface GigabitEthernet0/1/0/3.100
 ipv4 address 10.0.2.1 255.255.255.0
 ipv6 address 2001:db9:0:1::1/64
 dot1q vlan 100
!

RP/0/RP0/CPU0:router#
```

**sh int gig 0/1/0/3.100 | i MTU**

```
  MTU
```

**2004**

```
 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router#show im database interface gigabitEthernet 0/1/0/3.100

View: OWN - Owner, L3P - Local 3rd Party, G3P - Global 3rd Party,
      LDP - Local Data Plane, GDP - Global Data Plane, RED - Redundancy

Node 0/1/CPU0 (0x11)

Interface GigabitEthernet0/1/0/3.100, ifh 0x01180260 (up, 2004)
  Interface flags:          0x0000000000000597 (IFINDEX|SUP_NAMED_SUB
                            |BROADCAST|CONFIG|VIS|DATA|CONTROL)
  Encapsulation:            dot1q
  Interface type:           IFT_VLAN_SUBIF
  Control parent:           GigabitEthernet0/1/0/3
  Data parent:              GigabitEthernet0/1/0/3
  Views:                    GDP|LDP|L3P|OWN

  Protocol        Caps (state, mtu)
  --------        -----------------
  None            vlan_jump (up, 2004)
  None            dot1q (up, 2004)
  arp             arp (up, 1986)
  ipv4            ipv4 (up,
```

**1986**

```
)
  ipv6            ipv6_preswitch (up, 1986)
  ipv6            ipv6 (down,
```

**1986**

```
)

RP/0/RP0/CPU0:router#
```

In the **show** commands, the MTU of the sub-interface is 2004; add 4 bytes to the MTU of the main interface because there is one dot1q tag configured under the sub-interface.

However, the MTU of the IPv4 and IPv6 packets are still the same as that of the main interface (1986). This is because the MTU of the L3 protocols is now computed as: 2004 - 14 - 4 = 1986.

The **mtu** command can be configured under the sub-interface, but the configured MTU is applied only if it is lower or equal to the MTU that is inherited from the main interface (4 bytes larger than the MTU of the main interface).

When the MTU of the sub-interface that is larger than the inherited MTU, it is not applied, as shown here:

```
<#root>

RP/0/RP0/CPU0:router#

sh int gig 0/1/0/3.100 | i MTU


  MTU 2004 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router#conf
RP/0/RP0/CPU0:router(config)#int gig 0/1/0/3.100
RP/0/RP0/CPU0:router(config-subif)#mtu 2100
RP/0/RP0/CPU0:router(config-subif)#commit
RP/0/RP0/CPU0:router(config-subif)#end
RP/0/RP0/CPU0:router#sh int gig 0/1/0/3.100 | i MTU
  MTU 2004 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router#
```

Thus, you can use only the **mtu** command in order to lower the MTU value inherited from the main interface.

Similarly, you can also use the MTU commands of L3 protocols (IPv4, IPv6, MPLS) in order to lower the value of the L3 MTU inherited from the sub-interface L2 payload. The L3 protocol MTU does not take effect when it is configured to a value that does not fit in the payload of the L2 MTU.

## L2VPN L2 Interface

The MTU for an L2VPN is important because Label Distribution Protocol (LDP) does not bring a pseudowire (PW) up when the MTUs on the attachment circuits at each side of a PW are not the same.

Here is a **show** command that illustrates that an L2VPN PW stays down when there is an MTU mismatch:

```
<#root>

RP/0/RP0/CPU0:router1#

sh l2vpn xconnect


Legend: ST = State, UP = Up, DN = Down, AD = Admin Down, UR = Unresolved,
        SB = Standby, SR = Standby Ready, (PP) = Partially Programmed

XConnect                    Segment 1                       Segment 2
Group      Name       ST    Description            ST       Description            ST
------------------------    ----------------------------    ----------------------------
```

```
mtu        mtu
```

**DN**

```
   Gi0/0/0/2.201          UP       10.0.0.12       201
```

**DN**

```
--------------------------------------------------------------------------------
RP/0/RP0/CPU0:router1#sh l2vpn xconnect detail

Group mtu, XC mtu, state is down; Interworking none
  AC: GigabitEthernet0/0/0/2.201, state is up
    Type VLAN; Num Ranges: 1
    VLAN ranges: [201, 201]
    MTU 2000; XC ID 0x1080001; interworking none
    Statistics:
      packets: received 0, sent 0
      bytes: received 0, sent 0
      drops: illegal VLAN 0, illegal length 0
  PW: neighbor 10.0.0.12, PW ID 201, state is
```

**down**

```
 ( local ready )
    PW class mtu-class, XC ID 0xfffe0001
    Encapsulation MPLS, protocol LDP
    Source address 10.0.0.2
    PW type Ethernet, control word disabled, interworking none
    PW backup disable delay 0 sec
    Sequencing not set

    PW Status TLV in use
      MPLS         Local                          Remote
      ------------ ------------------------------ ------------------------------
      Label        16046                          16046
      Group ID     0x1080100                      0x6000180
      Interface    GigabitEthernet0/0/0/2.201     GigabitEthernet0/1/0/3.201
      MTU
```

**2000**

**1986**

```
      Control word disabled                       disabled
      PW type      Ethernet                       Ethernet
      VCCV CV type 0x2                            0x2
                   (LSP ping verification)        (LSP ping verification)
      VCCV CC type 0x6                            0x6
                   (router alert label)           (router alert label)
                   (TTL expiry)                   (TTL expiry)
      ------------ ------------------------------ ------------------------------
    Incoming Status (PW Status TLV):
      Status code: 0x0 (Up) in Notification message
    Outgoing Status (PW Status TLV):
      Status code: 0x0 (Up) in Notification message
    MIB cpwVcIndex: 4294836225
    Create time: 18/04/2013 16:20:35 (00:00:37 ago)
    Last time status changed: 18/04/2013 16:20:43 (00:00:29 ago)
    Error:
```

```
MTU mismatched


    Statistics:
      packets: received 0, sent 0
      bytes: received 0, sent 0
RP/0/RP0/CPU0:router1#

RP/0/RP0/CPU0:router1#
sh int GigabitEthernet0/0/0/2 | i MTU


  MTU 2014 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)

RP/0/RP0/CPU0:router1#
sh int GigabitEthernet0/0/0/2.201 | i MTU


  MTU 2018 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router1#
```

In this example, note that the MPLS L2VPN provider edges (PEs) at each side must signal the same MTU value in order to bring the PW up.

The MTU signaled by MPLS LDP does not include the L2 overhead. This is different from the XR interface **config** and **show** commands that include the L2 overhead. The MTU on the sub-interface is 2018 bytes (as inherited from the main interface of 2014 bytes), but LDP signaled an MTU of 2000 bytes. As a result, it subtracts 18 bytes (14 bytes of Ethernet header + 4 bytes of 1 dot1q tag) from the L2 header.

It is important to understand how each device computes the MTU values of the attachment circuits in order to fix MTU mismatches. This depends upon parameters such as vendor, platform, software version, and configuration.

### EVC (ASR9000)

The Cisco ASR 9000 Series Aggregation Services Router uses the EVC infrastructure model, which allows flexible VLAN matching on L2VPN L2 interfaces and sub-interfaces.

The EVC L2VPN L2 interfaces have these characteristics:

- They allow configuration of one or more tags with the **encapsulation** command.
- By default and with just the **encapsulation** command, tags are preserved and transported over PWs. As a result, you do not need to strip tags by default, as you need to do on non-EVC platforms.
- Use the **rewrite** command when you decide to pop the incoming tags or to push some additional tags on top of the incoming frame.

In order to compute the sub-interface MTU, take the main interface MTU (either the default or the one configured manually under the main interface), and add 4 bytes for each VLAN tag configured with the **encapsulation** command.

When there is an **mtu** command under the sub-interface, it takes effect only if it is lower than the computed MTU. The **rewrite** command does not influence the MTU of the sub-interface.

Here is an example:

<#root>

```
RP/0/RSP0/CPU0:router2#
```

**sh run int gig 0/1/0/3**

```
interface GigabitEthernet0/1/0/3
 cdp
 mtu 2014
 negotiation auto
!

RP/0/RSP0/CPU0:router2#
```

**sh run int gig 0/1/0/3.201**

```
interface GigabitEthernet0/1/0/3.201 l2transport
 encapsulation dot1q 201 second-dot1q 10
 rewrite ingress tag pop 2 symmetric
!

RP/0/RSP0/CPU0:router2#
RP/0/RSP0/CPU0:router2#
```

**sh int gig 0/1/0/3.201**

```
GigabitEthernet0/1/0/3.201 is up, line protocol is up
  Interface state transitions: 1
  Hardware is VLAN sub-interface(s), address is 0024.986c.63f3
  Layer 2 Transport Mode
  MTU 2022 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
```

In this example, the MTU in the main interface is 2014 bytes; add 8 bytes because there are two tags configured under the sub-interface.

If you configure **mtu 2026** bytes under the sub-interface, it is not applied because it is larger than the MTU of the sub-interface inherited from the main interface (2022). As a result, you can configure only a sub-interface MTU lower than 2022 bytes.

Based on this sub-interface MTU, compute the MTU of the MPLS LDP payload that is signaled to the neighbor, and make sure that it is identical to the one computed by the remote L2VPN PE. This is where the **rewrite** command comes into play.

In order to compute the MTU of the MPLS LDP payload, take the MTU of the sub-interface, then:

1. Subtract 14 bytes for the Ethernet header.
2. Subtract 4 bytes for each tag popped in the **rewrite** command configured under the sub-interface.
3. Add 4 bytes for each tag pushed in the **rewrite** command configured under the sub-interface.

This is the same example with the QinQ configuration on gig 0/1/0/3.201:

<#root>

```
interface GigabitEthernet0/1/0/3
 cdp
 mtu 2014
 negotiation auto
```

```
!
interface GigabitEthernet0/1/0/3.201 l2transport
 encapsulation dot1q 201 second-dot1q 10
 rewrite ingress tag pop 2 symmetric
!

RP/0/RSP0/CPU0:router2#
sh int gig 0/1/0/3.201


GigabitEthernet0/1/0/3.201 is up, line protocol is up
  Interface state transitions: 1
  Hardware is VLAN sub-interface(s), address is 0024.986c.63f3
  Layer 2 Transport Mode
  MTU 2022 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
```

This is the computation for the MTU of the MPLS LDP payload:

1. MTU value of the sub-interface MTU: 2022 bytes
2. Subtract 14 bytes of Ethernet header: 2022 - 14 = 2008 bytes
3. Subtract 4 bytes for each popped tag in the **rewrite** command: 2008 - 4 * 2 = 2000

Ensure that the remote side advertises an MPLS LDP payload of 2000 bytes. Otherwise, adjust the local or remote attachment circuit (AC) MTU size so they match.

```
<#root>

RP/0/RSP0/CPU0:router2#
sh l2vpn xconnect det


Group mtu, XC mtu, state is up; Interworking none
  AC: GigabitEthernet0/1/0/3.201, state is up
    Type VLAN; Num Ranges: 1
    Outer Tag: 201
    VLAN ranges: [10, 10]
    MTU 2000; XC ID 0x1880003; interworking none
```

#### Specific Ethernet Flow Point (EFP) Encapsulation Commands

These encapsulations count as matching zero tags, so they do not increase the sub-interface MTU:

- **encapsulation untagged**
- **encapsulation default**

These encapsulation modifiers do not affect the number of tags required in order to compute the sub-interface MTU:

- native
- payload-ethertype
- exact
- cos

- ingress source-mac or ingress destination-mac

**encapsulation [dot1q|dot1ad] priority-tagged** counts as matching a single tag.

The 'any' keyword used as the innermost tag match does not increase the sub-interface MTU.

- **encapsulation dot1q any** does not increase the sub-interface MTU.
- **encapsulation dot1ad 10 dot1q any** is accounted as one tag; it increases the sub-interface MTU by 4 bytes.
- **encapsulation dot1ad any dot1q 7** is accounted as two tags; it increases the sub-interface MTU by 8 bytes.

The ranges of VLAN-IDs increment the sub-interface MTU:

- **encapsulation dot1q 10-100** is accounted as one tag; it increases the sub-interface MTU by 4 bytes.

The encapsulation MTU overhead of an EFP that is a disjunctive match is treated as the MTU of its highest element.

- **encapsulation dot1q 10-100, untagged** is accounted as one tag because the range 10 -100 is the highest element.

**Non EVC (XR 12000 and CRS)**

Routers like the Cisco XR 12000 Series router and the Carrier Routing System (CRS) use the traditional configuration for the VLAN matching on sub-interfaces. These characteristics apply to L2VPN L2 interfaces on CRS and on XR 12000 routers that do not follow the EVC model:

- On non-EVC platforms, the incoming dot1q or dot1ad tags are automatically stripped when they are received on an L2 transport sub-interface.
- When you are computing the payload size for MPLS LDP to signal, subtract the size of the tags from the MTU of the sub-interface, as seen in the **show interface** command.
- This is similar to the case of a routed sub-interface.
- The sub-interface inherits its MTU from the main interface; add the 4 bytes for each tag to the MTU of the main interface in order to compute the MTU of the sub-interface. For instance, if a QinQ sub-interface has 2 dot1q tags, the sub-interface, by default, has a MTU that is 8 bytes larger than the MTU of the main interface.
- You can also use the **mtu** command under the sub-interface, but it is used only to reduce the MTU of the sub-interface, which is inherited from the MTU of the main interface.

Here are several examples that illustrate these characteristics.

This example shows how a non-EVC sub-interface is configured:

<#root>

```
RP/0/RP0/CPU0:router1#
```

```
sh run int gigabitEthernet 0/0/0/2.201
```

```
interface GigabitEthernet0/0/0/2.201 l2transport
 dot1q vlan 201
!

RP/0/RP0/CPU0:router1#
```

The non-EVC platforms use the **dot1q vlan** or **dot1ad vlan** commands instead of the **encapsulation** and **rewrite** commands of the EVC platforms (ASR9000).

If you do not configure an MTU explicitly on the main or sub-interface, then a 1500 byte L3 packet can be received by default:

```
<#root>

RP/0/RP0/CPU0:router1#

sh int gig 0/0/0/2 | i MTU


  MTU 1514 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router1#

sh int gig 0/0/0/2.201 | i MTU


  MTU 1518 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router1#
```

The sub-interface MTU is computed from the main interface MTU (1514); add 4 bytes for each dot1q tag. Because there is one tag configured on the sub-interface with the **dot1q vlan 201** command, add 4 bytes to 1514 for an MTU of 1518 bytes.

The corresponding payload MTU in MPLS LDP is 1500 bytes, since the 14 bytes of Ethernet header are not counted and the one dot1q tag is popped automatically by the non-EVC platform when it goes over the PW:

```
<#root>

RP/0/RP0/CPU0:router1#

sh l2vpn xconnect detail



Group mtu, XC mtu, state is down; Interworking none
  AC: GigabitEthernet0/0/0/2.201, state is up
    Type VLAN; Num Ranges: 1
    VLAN ranges: [201, 201]
    MTU 1500; XC ID 0x1080001; interworking none
```

If you increase the MTU of the main interface to 2014 bytes, the MTU of the sub-interface is increased accordingly:

```
<#root>

RP/0/RP0/CPU0:router1#

sh run int gig 0/0/0/2


interface GigabitEthernet0/0/0/2
```

```
 description static lab connection to head 4/0/0 - dont change
 cdp
 mtu 2014
 ipv4 address 10.0.100.1 255.255.255.252
 load-interval 30
!

RP/0/RP0/CPU0:router1#
```

**sh run int gig 0/0/0/2.201**

```
interface GigabitEthernet0/0/0/2.201 l2transport
 dot1q vlan 201
!

RP/0/RP0/CPU0:router1#
```

**sh int gig 0/0/0/2 | i MTU**

```
  MTU 2014 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router1#
```

**sh int gig 0/0/0/2.201 | i MTU**

```
  MTU 2018 bytes, BW 1000000 Kbit (Max: 1000000 Kbit)
RP/0/RP0/CPU0:router1#
```

**sh l2vpn xconnect detail**

```
Group mtu, XC mtu, state is down; Interworking none
  AC: GigabitEthernet0/0/0/2.201, state is up
    Type VLAN; Num Ranges: 1
    VLAN ranges: [201, 201]
    MTU 2000; XC ID 0x1080001; interworking none
```

So, in order to compute the MPLS LDP MTU, subtract 14 bytes of Ethernet header, and add 4 bytes for each tag configured under the sub-interface.

# Automatic Ethernet Interface Driver MTU and MRU Configuration

On Ethernet interfaces the interface driver is configured with an MTU and MRU that is based on the interface MTU configuration.

The configured MTU and MRU on the Ethernet interface driver can be seen with the **show controller <interface> all** command.

In releases earlier than Cisco IOS XR Release 5.1.1, the MTU and MRU on the Ethernet interface driver were automatically configured based on the Cisco IOS XR MTU configuration on the interface.

The MTU/MRU configured on the Ethernet driver was simply based on the configured MTU + 12 bytes for the addition of 2 Ethernet Tags and the CRC field. The 12 bytes were added to the Ethernet driver MTU/MRU regardless of if there were any VLAN tags configured on the sub-interfaces.

An example with all Cisco IOS XR versions earlier than Cisco IOS XR Release 5.1.1 and a default MTU of 1514 on an ASR 9000 interface is shown here:

```
<#root>

RP/0/RSP0/CPU0:ASR2#

show interface Gi0/2/0/0


GigabitEthernet0/2/0/0 is up, line protocol is up
  Interface state transitions: 3
  Hardware is GigabitEthernet, address is 18ef.63e2.0598 (bia 18ef.63e2.0598)
  Description: Static_Connections_to_ME3400-1_Gi_0_2 - Do Not Change
  Internet address is Unknown


MTU 1514 bytes

, BW 1000000 Kbit (Max: 1000000 Kbit)
<snip>

MTU/MRU programmed on ethernet interface driver is 1514 + 12 bytes

RP/0/RSP0/CPU0:ASR2#show controllers Gi0/2/0/0  all

<snip>
Operational values:
    Speed: 1Gbps
    Duplex: Full Duplex
    Flowcontrol: None
    Loopback: None (or external)


MTU: 1526



    MRU: 1526


    Inter-packet gap: standard (12)
<snip>
```

In Cisco IOS XR Release 5.1.1 and later, the MTU and MRU that is used on the Ethernet interface driver has changed and is now based on the number of VLAN tags that are configured on any of the sub-interfaces.

If no VLAN tags are configured on any sub-interfaces, the driver MTU/MRU equals the configured MTU on interface + 4 CRC bytes, for example 1514 + 4 = 1518 bytes.

If one VLAN is configured on any sub-interfaces, the driver MTU/MRU equals the configured MTU + 8 bytes (1 tag + CRC), for example 1514 + 8 = 1522 bytes.

If two VLAN tags are configured on any sub-interfaces, the driver MTU/MRU equals the configured MTU + 12 bytes (2 tags + CRC), for example 1514 + 12 = 1526 bytes

If QinQ with the **any** keyword is configured for the second-do1q tag the driver MTU/MRU equals the

configured MTU + 8 bytes (1 tag + CRC), for example 1514 + 8 = 1522 bytes.

These examples display the behavior in Cisco IOS XR Release 5.1.1 and later on an ASR 9000:

```
<#root>

RP/0/RSP0/CPU0:ASR2#

sh run int ten0/1/0/0


interface TenGigE0/1/0/0
 cdp

RP/0/RSP0/CPU0:ASR2#

show controllers ten0/1/0/0 all



<snip>
Operational values:
    Speed: 10Gbps
    Duplex: Full Duplex
    Flowcontrol: None
    Loopback: Internal


MTU: 1518



    MRU: 1518


    Inter-packet gap: standard (12)
<snip>

RP/0/RSP0/CPU0:ASR2#

config


RP/0/RSP0/CPU0:ASR2(config-if)#

int ten0/1/0/0.1


RP/0/RSP0/CPU0:ASR2(config-subif)#

encapsulation dot1q 1


RP/0/RSP0/CPU0:ASR2(config-subif)#

commit



RP/0/RSP0/CPU0:ASR2#

show controllers ten0/1/0/0  all
```

```
<snip>
Operational values:
    Speed: 10Gbps
    Duplex: Full Duplex
    Flowcontrol: None
    Loopback: Internal

MTU: 1522

    MRU: 1522

    Inter-packet gap: standard (12)
<snip>

RP/0/RSP0/CPU0:ASR2#config
RP/0/RSP0/CPU0:ASR2(config)#
int ten0/1/0/0.2

RP/0/RSP0/CPU0:ASR2(config-subif)#
encapsulation dot1q 10 second-dot1q 20

RP/0/RSP0/CPU0:ASR2(config-subif)#
commit

RP/0/RSP0/CPU0:ASR2#
show controllers ten0/1/0/0 all

<snip>
Operational values:
    Speed: 10Gbps
    Duplex: Full Duplex
    Flowcontrol: None
    Loopback: Internal

MTU: 1526

    MRU: 1526

    Inter-packet gap: standard (12)
<snip>

RP/0/RSP0/CPU0:ASR2#config
RP/0/RSP0/CPU0:ASR2(config)#
int ten0/2/0/0

RP/0/RSP0/CPU0:ASR2(config)#
```

```
cdp


RP/0/RSP0/CPU0:ASR2(config)#
int ten0/2/0/0.1 l2transport


RP/0/RSP0/CPU0:ASR2(config-subif)#
encapsulation dot1q 10 second-dot1q any


RP/0/RSP0/CPU0:ASR2(config-subif)#
commit



RP/0/RSP0/CPU0:ASR2#
show controllers ten0/1/0/0 all



<snip>
Operational values:
    Speed: 10Gbps
    Duplex: Full Duplex
    Flowcontrol: None
    Loopback: Internal


MTU: 1522



    MRU: 1522


    Inter-packet gap: standard (12)
<snip>
```

In most situations this behavior change in Release 5.1.1 and later should not require any changes to the configuration of the MTU on the interface.

This behavior change can cause problems in the case of a sub-interface configured with a single VLAN tag, but receives packets with two VLAN tags. In that situation the packets received can exceed the MRU on the Ethernet interface driver. In order to eliminate that condition, the interface MTU can either be increased by 4 bytes or the sub-interface configured with two VLAN tags.

The automatic Ethernet interface driver MTU and MRU configuration in Release 5.1.1 behavior is the same for a CRS and ASR 9000 routers. But a CRS router that runs Release 5.1.1 does not include the 4 byte CRC in the MTU and MRU value displayed in the **show controller** output. The behavior of how it is reported is not the same between CRS and ASR9000.

```
<#root>

RP/0/RP0/CPU0:CRS#
sh run int ten0/4/0/0
```

```
Mon May 19 08:49:26.109 UTC
interface TenGigE0/4/0/0

<snip>
Operational values:
   Speed: 10Gbps
   Duplex: Full Duplex
   Flowcontrol: None
   Loopback: None (or external)
   MTU: 1514
   MRU: 1514
   Inter-packet gap: standard (12)

RP/0/RP0/CPU0:CRS(config)#
int ten0/4/0/0.1


RP/0/RP0/CPU0:CRS(config-subif)#
encapsulation dot1q 1


RP/0/RP0/CPU0:CRS(config-subif)#
commit



Operational values:
   Speed: 10Gbps
   Duplex: Full Duplex
   Flowcontrol: None
   Loopback: None (or external)
   MTU: 1518
   MRU: 1518
   Inter-packet gap: standard (12)
```

The way the MTU and MRU is displayed in the show controller output on the ASR 9000 will change in the future so that the 4 bytes of CRC will not be included in the MTU/MRU value displayed. This future change can be tracked with Cisco bug ID CSCuo93379.

## Convert the Configuration When You Upgrade from a Release Earlier than Release 5.1.1 to Release 5.1.1 or Later

- Default MTU:

If there was a main interface without any sub-interface and without any **mtu** command in a release earlier than Release 5.1.1:

```
interface TenGigE0/1/0/19
 l2transport
 !
!
```

And this interface transports dot1q or QinQ frames, then the MTU should be manually configured to "mtu 1522" in Release 5.1.1 and later:

```
interface TenGigE0/1/0/19
 mtu 1522
 l2transport
 !
!
```

This configuration allows QinQ frames to be transported as in the earlier releases. The MTU value could be configured to 1518 if only dot1q and not QinQ are to be transported.

If there were sub-interfaces configured for dot1q or QinQ, but with the "any" keyword and no QinQ sub-interfaces with 2 explicit tags was configured in a release earlier than Release 5.1.1:

```
interface TenGigE0/1/0/19
!
interface TenGigE0/1/0/19.100 l2transport
 encapsulation dot1q 100
!
interface TenGigE0/1/0/19.101 l2transport
  encapsulation dot1q 101 second-dot1q any
!
```

This configuration in Release 5.1.1 and later will only allow to transport frames with one tag so the MTU should also be increased manually by 4 bytes if QinQ frames are to be transported:

```
interface TenGigE0/1/0/19
 mtu 1518
!
interface TenGigE0/1/0/19.100 l2transport
 encapsulation dot1q 100
!
interface TenGigE0/1/0/19.101 l2transport
 encapsulation dot1q 101 second-dot1q any
!
```

If a QinQ sub-interface with 2 explicit tags (that do not use the "any" keyword) is configured, there is no need to modify the MTU configuration when you upgrade to Release 5.1.1 and later:

```
interface TenGigE0/1/0/19
!
interface TenGigE0/1/0/19.101 l2transport
 encapsulation dot1q 101 second-dot1q 200
!
```

If there is no L2 transport sub-interface but only L3 routed interfaces, it is expected that the MTU configuration would match at both sides and there would not be frames larger than the MTU that is transported. There is no need to update the MTU configuration when you upgrade to Release 5.1.1 and later.

- Non-default MTU in release earlier than Release 5.1.1:

Similarly, when a non-default MTU was configured in a release earlier than Release 5.1.1 and no sub-interface was configured and dot1q or QinQ frames have to be transported, then the configured MTU value should be increased by 8 bytes when you upgrade to Release 5.1.1 or later.

Release earlier than Release 5.1.1:

```
interface TenGigE0/1/0/19
 mtu 2000
 l2transport
 !
!
```

The MTU should be manually increased by 8 bytes when you upgrade to Release 5.1.1 and later:

```
interface TenGigE0/1/0/19
 mtu 2008
 l2transport
 !
!
```

The configured MTU value should also be increased by 4 bytes if there is a dot1q sub-interface and no QinQ sub-interface or a QinQ sub-interface with the any keyword for the second-dot1q tag.

Release earlier than Release 5.1.1:

```
interface TenGigE0/1/0/19
 mtu 2000
!
interface TenGigE0/1/0/19.100 l2transport
 encapsulation dot1q 100
!
interface TenGigE0/1/0/19.101 l2transport
 encapsulation dot1q 101 second-dot1q any
!
```

Release 5.1.1 and later:

```
interface TenGigE0/1/0/19
```

```
 mtu 2004
!
interface TenGigE0/1/0/19.100 l2transport
 encapsulation dot1q 100
!
interface TenGigE0/1/0/19.101 l2transport
 encapsulation dot1q 101 second-dot1q any
!
```

If a QinQ sub-interface with 2 explicit tags (that do not use the "any" keyword) is configured, there is no need to modify the MTU configuration when you upgrade to Release 5.1.1 and later.

```
interface TenGigE0/1/0/19
!
interface TenGigE0/1/0/19.101 l2transport
 encapsulation dot1q 101 second-dot1q 200
!
```

If there is no L2 transport sub-interface, but only L3 routed interfaces, it is expected that the MTU configuration would match at both sides and there would not be frames larger than the MTU that is transported. There is no need to update the MTU configuration when you upgrade to Release 5.1.1 and later.