Implementing Tunnels

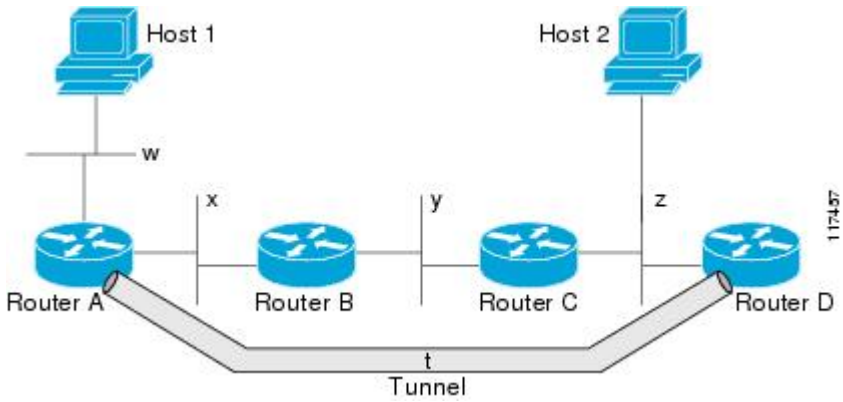
This module describes the various types of tunneling techniques. Configuration details and examples are provided for the tunnel types that use physical or virtual interfaces. Many tunneling techniques are implemented using technology-specific commands, and links are provided to the appropriate technology modules.

Tunneling provides a way to encapsulate arbitrary packets inside a transport protocol. Tunnels are implemented as virtual interfaces to provide a simple interface for configuration purposes. The tunnel interface is not tied to specific "passenger" or "transport" protocols, but rather is an architecture to provide the services necessary to implement any standard point-to-point encapsulation scheme.

Restrictions for Implementing Tunnels

- It is important to allow the tunnel protocol to pass through a firewall and access control list (ACL) check.
- Multiple point-to-point tunnels can saturate the physical link with routing information if the bandwidth is not configured correctly on a tunnel interface.
- A tunnel looks like a single hop link, and routing protocols may prefer a tunnel over a multihop physical path. The tunnel, despite looking like a single hop link, may traverse a slower path than a multihop link. A tunnel is as robust and fast, or as unreliable and slow, as the links that it actually traverses. Routing protocols that make their decisions based only on hop counts will often prefer a tunnel over a set of physical links. A tunnel might appear to be a one-hop, point-to-point link and have the lowest-cost path, but the tunnel may actually cost more in terms of latency when compared to an alternative physical topology. For example, in the topology shown in the figure below, packets from Host 1 will appear to travel across networks w, t, and z to get to Host 2 instead of taking the path w, x, y, and z because the tunnel hop count appears shorter. In fact, the packets going through the tunnel will still be traveling across Router A, B, and C, but they must also travel to Router D before coming back to Router C.

Figure 1: Tunnel Precautions: Hop Counts



- A tunnel may have a recursive routing problem if routing is not configured accurately. The best path to a tunnel destination is via the tunnel itself; therefore recursive routing causes the tunnel interface to flap. To avoid recursive routing problems, keep the control-plane routing separate from the tunnel routing by using the following methods:
  - Use a different autonomous system number or tag.
  - Use a different routing protocol.
  - Ensure that static routes are used to override the first hop (watch for routing loops).

  The following error is displayed when there is recursive routing to a tunnel destination:

```
%TUN-RECURDOWN Interface Tunnel 0
temporarily disabled due to recursive routing
```

Information About Implementing Tunnels

Tunneling Versus Encapsulation

To understand how tunnels work, you must be able to distinguish between concepts of encapsulation and tunneling. Encapsulation is the process of adding headers to data at each layer of a particular protocol stack. The Open Systems Interconnection (OSI) reference model describes the functions of a network. To send a data packet from one host (for example, a PC) to another on a network, encapsulation is used to add a header in front of the data packet at each layer of the protocol stack in descending order. The header must contain a data field that indicates the type of data encapsulated at the layer immediately above the current layer. As the packet ascends the protocol stack on the receiving side of the network, each encapsulation header is removed in reverse order.

Tunneling encapsulates data packets from one protocol within a different protocol and transports the packets on a foreign network. Unlike encapsulation, tunneling allows a lower-layer protocol and a same-layer protocol to be carried through the tunnel. A tunnel interface is a virtual (or logical) interface. Tunneling consists of three main components:

- Passenger protocol—The protocol that you are encapsulating. For example, IPv4 and IPv6 protocols.
- Carrier protocol—The protocol that encapsulates. For example, generic routing encapsulation (GRE) and Multiprotocol Label Switching (MPLS).

- Transport protocol--The protocol that carries the encapsulated protocol. The main transport protocol is IP.

Tunnel ToS

Tunnel type of service (ToS) allows you to tunnel network traffic and group all packets in the same ToS byte value. The ToS byte values and Time-to-Live (TTL) hop-count value can be set in the encapsulating IP header of tunnel packets for an IP tunnel interface on a router. Tunnel ToS feature is supported for Cisco Express Forwarding (formerly known as CEF), fast switching, and process switching.

The ToS and TTL byte values are defined in RFC 791. RFC 2474, and RFC 2780 obsolete the use of the ToS byte as defined in RFC 791. RFC 791 specifies that bits 6 and 7 of the ToS byte (the first two least significant bits) are reserved for future use and should be set to 0.

EoMPLS over GRE

Ethernet over MPLS (EoMPLS) is a tunneling mechanism that allows you to tunnel Layer 2 traffic through a Layer 3 MPLS network. EoMPLS is also known as Layer 2 tunneling.

EoMPLS effectively facilitates Layer 2 extension over long distances. EoMPLS over GRE helps you to create the GRE tunnel as hardware-based switched, and encapsulates EoMPLS frames within the GRE tunnel. The GRE connection is established between the two core routers, and then the MPLS label switched path (LSP) is tunneled over.

GRE encapsulation is used to define a packet that has header information added to it prior to being forwarded. De-encapsulation is the process of removing the additional header information when the packet reaches the destination tunnel endpoint.

When a packet is forwarded through a GRE tunnel, two new headers are added to the front of the packet and hence the context of the new payload changes. After encapsulation, what was originally the data payload and separate IP header are now known as the GRE payload. A GRE header is added to the packet to provide information on the protocol type and the recalculated checksum. A new IP header is also added to the front of the GRE header. This IP header contains the destination IP address of the tunnel.

The GRE header is added to packets such as IP, Layer 2 VPN, and Layer 3 VPN before the header enters into the tunnel. All routers along the path that receives the encapsulated packet use the new IP header to determine how the packet can reach the tunnel endpoint.

In IP forwarding, on reaching the tunnel destination endpoint, the new IP header and the GRE header are removed from the packet and the original IP header is used to forward the packet to the final destination.

The EoMPLS over GRE feature removes the new IP header and GRE header from the packet at the tunnel destination, and the MPLS label is used to forward the packet to the appropriate Layer 2 attachment circuit or Layer 3 VRF.

The scenarios in the following sections describe the L2VPN and L3VPN over GRE deployment on provider edge (PE) or provider (P) routers:

Provider Edge to Provider Edge Generic Routing EncapsulationTunnels

In the Provider Edge to Provider Edge (PE) GRE tunnels scenario, a customer does not transition any part of the core to MPLS but prefers to offer EoMPLS and basic MPLS VPN services. Therefore, GRE tunneling of MPLS traffic is done between PEs.

Provider to Provider Generic Routing Encapsulation Tunnels

In the Provider to Provider (P) GRE tunnels scenario, Multiprotocol Label Switching (MPLS) is enabled between Provider Edge (PE) and P routers but the network core can either have non-MPLS aware routers or IP encryption boxes. In this scenario, GRE tunneling of the MPLS labeled packets is done between P routers.

Provider Edge to Provider Generic Routing Encapsulation Tunnels

In a Provider Edge to Provider GRE tunnels scenario, a network has MPLS-aware P to P nodes. GRE tunneling is done between a PE to P non-MPLS network segment.

Features Specific to Generic Routing Encapsulation

You should understand the following configurations and information for a deployment scenario:

- Tunnel endpoints can be loopbacks or physical interfaces.
- Configurable tunnel keepalive timer parameters per endpoint and a syslog message must be generated when the keepalive timer expires.
- Bidirectional forwarding detection (BFD) is supported for tunnel failures and for the Interior Gateway Protocol (IGP) that use tunnels.
- IGP load sharing across a GRE tunnel is supported.
- IGP redundancy across a GRE tunnel is supported.
- Fragmentation across a GRE tunnel is supported.
- Ability to pass jumbo frames is supported.
- All IGP control plane traffic is supported.
- IP ToS preservation across tunnels is supported.
- A tunnel should be independent of the endpoint physical interface type; for example, ATM, Gigabit, Packet over SONET (POS), and TenGigabit.
- Up to 100 GRE tunnels are supported.

Features Specific to Ethernet over MPLS

- Any Transport over MPLS (AToM) sequencing.
- IGP load sharing and redundancy.
- Port mode Ethernet over MPLS (EoMPLS).
- Pseudowire redundancy.
- Support for up to to 200 EoMPLS virtual circuits (VCs).
- Tunnel selection and the ability to map a specific pseudowire to a GRE tunnel.
- VLAN mode EoMPLS.

Features Specific to Multiprotocol Label Switching Virtual Private Network

- Support for the PE role with IPv4 VRF.
- Support for all PE to customer edge (CE) protocols.
- Load sharing through multiple tunnels and also equal cost IGP paths with a single tunnel.
- Support for redundancy through unequal cost IGP paths with a single tunnel.
- Support for the IP precedence value being copied onto the expression (EXP) bits field of the Multiprotocol Label Switching (MPLS) label and then onto the precedence bits on the outer IPv4 ToS field of the generic routing encapsulation (GRE) packet.

See the section, Example: Configuring EoMPLS over GRE, on page 16" for a sample configuration sequence of EoMPLS over GRE. For more details on EoMPLS over GRE, see the Deploying and Configuring MPLS Virtual Private Networks In IP Tunnel Environments document.

Path MTU Discovery

Path MTU Discovery (PMTUD) can be enabled on a GRE or IP-in-IP tunnel interface. When PMTUD (RFC 1191) is enabled on a tunnel interface, the router performs PMTUD processing for the GRE (or IP-in-IP) tunnel IP packets. The router always performs PMTUD processing on the original data IP packets that enter the tunnel. When PMTUD is enabled, packet fragmentation is not permitted for packets that traverse the tunnel because the Don't Fragment (DF) bit is set on all the packets. If a packet that enters the tunnel encounters a link with a smaller MTU, the packet is dropped and an Internet Control Message Protocol (ICMP) message is sent back to the sender of the packet. This message indicates that fragmentation was required (but not permitted) and provides the MTU of the link that caused the packet to be dropped.

Note PMTUD on a tunnel interface requires that the tunnel endpoint be able to receive ICMP messages generated by routers in the path of the tunnel. Ensure that ICMP messages can be received before using PMTUD over firewall connections.

Use the **tunnel path-mtu-discovery** command to enable PMTUD for the tunnel packets and use the **show interfaces tunnel** command to verify the tunnel PMTUD parameters. PMTUD works only on GRE and IP-in-IP tunnel interfaces.

QoS Options for Tunnels

A tunnel interface supports various quality of service (QoS) features as a physical interface. QoS provides a way to ensure that mission-critical traffic has an acceptable level of performance. QoS options for tunnels include support for applying generic traffic shaping (GTS) directly on the tunnel interface and support for class-based shaping using the modular QoS CLI (MQC). Tunnel interfaces also support class-based policing, but they do not support committed access rate (CAR).

GRE tunnels allow the router to copy the IP precedence bit values of the ToS byte to the tunnel or the GRE IP header that encapsulates the inner packet. Intermediate routers between the tunnel endpoints can use the IP precedence values to classify packets for QoS features such as policy routing, weighted fair queueing (WFQ), and weighted random early detection (WRED).

When packets are encapsulated by tunnel or encryption headers, QoS features are unable to examine the original packet headers and correctly classify the packets. Packets that travel across the same tunnel have the same tunnel headers, so the packets are treated identically if the physical interface is congested. Tunnel packets can, however, be classified before tunneling and encryption can occur when a user applies the QoS preclassify feature on the tunnel interface or on the crypto map.

Note Class-based WFQ (CBWFQ) inside class-based shaping is not supported on a multipoint interface.

For examples of how to implement some QoS features on a tunnel interface, see the sectionConfiguring QoS Options on Tunnel Interfaces Examples, on page 18.

How to Implement Tunnels

Determining the Tunnel Type

Before configuring a tunnel, you must determine the type of tunnel you want to create.

SUMMARY STEPS

1. Determine the passenger protocol. A passenger protocol is the protocol that you are encapsulating.
2. Determine the **tunnel mode** command keyword, if appropriate.

DETAILED STEPS

Step 1 Determine the passenger protocol. A passenger protocol is the protocol that you are encapsulating.

Step 2 Determine the **tunnel mode** command keyword, if appropriate.

The table below shows how to determine the appropriate keyword to be used with the **tunnel mode** command.

Table 1: Determining the tunnel mode Command Keyword

Keyword	Purpose
dvmrp	Use the **dvmrp** keyword to specify that the Distance Vector Multicast Routing Protocol encapsulation will be used.
gre ip	Use the **gre** and **ip** keywords to specify that GRE encapsulation over IP will be used.
gre ipv6	Use the **gre** and **ipv6** keywords to specify that GRE encapsulation over IPv6 will be used.
ipip [**decapsulate-any**]	Use the **ipip** keyword to specify that IP-in-IP encapsulation will be used. The optional **decapsulate-any** keyword terminates any number of IP-in-IP tunnels at one tunnel interface. Note that this tunnel will not carry any outbound traffic; however, any number of remote tunnel endpoints can use a tunnel configured as their destination.
ipv6	Use the **ipv6** keyword to specify that generic packet tunneling in IPv6 will be used.

Keyword	Purpose
ipv6ip	Use the **ipv6ip** keyword to specify that IPv6 will be used as the passenger protocol and IPv4 as both the carrier (encapsulation) and transport protocol. When additional keywords are not used, manual IPv6 tunnels are configured. Additional keywords can be used to specify IPv4-compatible, 6to4, or ISATAP tunnels.
mpls	Use the **mpls** keyword to specify that MPLS will be used for configuring traffic engineering (TE) tunnels.

Configuring an IPv4 GRE Tunnel

Perform this task to configure a GRE tunnel. A tunnel interface is used to pass protocol traffic across a network that does not normally support the protocol. To build a tunnel, you must define a tunnel interface on each of the two routers, and the tunnel interfaces must reference each other. At each router, the tunnel interface must be configured with a Layer 3 address. The tunnel endpoints, tunnel source, and tunnel destination must be defined, and the type of tunnel must be selected. Optional steps can be performed to customize the tunnel.

Remember to configure the router at each end of the tunnel. If only one side of a tunnel is configured, the tunnel interface may still come up and stay up (unless keepalive is configured), but packets going into the tunnel will be dropped.

GRE Tunnel Keepalive

Keepalive packets can be configured to be sent over IP-encapsulated GRE tunnels. You can specify the rate at which keepalives are sent and the number of times that a device will continue to send keepalive packets without a response before the interface becomes inactive. GRE keepalive packets may be sent from both sides of a tunnel or from just one side.

Before you begin

Ensure that the physical interface to be used as the tunnel source in this task is up and configured with the appropriate IP address. For hardware technical descriptions and information about installing interfaces, see the hardware installation and configuration publication for your product.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface** *type number*
4. **bandwidth** *kb/s*
5. **keepalive** [*period* [*retries*]]
6. **tunnel source** {*ip-address* | *interface-type interface-number*}
7. **tunnel destination** {*hostname* | *ip-address*}
8. **tunnel key** *key-number*
9. **tunnel mode gre** { **ip** | **multipoint**}
10. **ip mtu** *bytes*
11. **tunnel mpls-ip-only**

12. **ip tcp mss** *mss-value*
13. **tunnel path-mtu-discovery** [**age-timer** {*aging-mins* | **infinite**}]
14. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**interface** *type number* **Example:** `Router(config)# interface tunnel 0`	Specifies the interface type and number, and enters interface configuration mode. • To configure a tunnel, use **tunnel** for the *type* argument.
Step 4	**bandwidth** *kb/s* **Example:** `Router(config-if)# bandwidth 1000`	Sets the current bandwidth value for an interface and communicates it to higher-level protocols. • Specifies the tunnel bandwidth to be used to transmit packets. • Use the *kb/s* argument to set the bandwidth, in kilobits per second (kb/s). **Note** This is only a routing parameter; it does not affect the physical interface. The default bandwidth setting on a tunnel interface is 9.6 kb/s. You should set the bandwidth on a tunnel to an appropriate value.
Step 5	**keepalive** [*period* [*retries*]] **Example:** `Router(config-if)# keepalive 3 7`	(Optional) Specifies the number of times the device will continue to send keepalive packets without response before bringing the tunnel interface protocol down. • GRE keepalive packets may be configured either on only one side of the tunnel or on both. • If GRE keepalive is configured on both sides of the tunnel, the *period* and *retries* arguments can be different at each side of the link. **Note** This command is supported only on GRE point-to-point tunnels.

	Command or Action	Purpose
Step 6	**tunnel source** {*ip-address* | *interface-type interface-number*} **Example:** `Router(config-if)# tunnel source GigabitEthernet 0/0/0`	Configures the tunnel source. • Use the *ip-address* argument to specify the source IP address. • Use the *interface-type* and *interface-number* arguments to specify the interface to be used. **Note** The tunnel source IP address and destination IP addresses must be defined on two separate devices.
Step 7	**tunnel destination** {*hostname* | *ip-address*} **Example:** `Router(config-if)# tunnel destination 10.0.2.1`	Configures the tunnel destination. • Use the *hostname* argument to specify the name of the host destination. • Use the *ip-address* argument to specify the IP address of the host destination. **Note** The tunnel source and destination IP addresses must be defined on two separate devices.
Step 8	**tunnel key** *key-number* **Example:** `Router(config-if)# tunnel key 1000`	(Optional) Enables an ID key for a tunnel interface. • Use the *key-number* argument to identify a tunnel key that is carried in each packet. • Tunnel ID keys can be used as a form of weak security to prevent improper configuration or injection of packets from a foreign source. **Note** This command is supported only on GRE tunnel interfaces. We do not recommend relying on this key for security purposes.
Step 9	**tunnel mode gre** { **ip** | **multipoint**} **Example:** `Device(config-if)# tunnel mode gre ip`	Specifies the encapsulation protocol to be used in the tunnel. • Use the **gre ip** keywords to specify that GRE over IP encapsulation will be used. • Use the **gre multipoint** keywords to specify that multipoint GRE (mGRE) will be used.
Step 10	**ip mtu** *bytes* **Example:** `Device(config-if)# ip mtu 1400`	(Optional) Sets the MTU size of IP packets sent on an interface. • If an IP packet exceeds the MTU set for the interface, the Cisco software will fragment it unless the DF bit is set.

	Command or Action	Purpose
		• All devices on a physical medium must have the same protocol MTU in order to operate. • For IPv6 packets, use the **ipv6 mtu** command. **Note** If the **tunnel path-mtu-discovery** command is enabled do not configure this command.
Step 11	**tunnel mpls-ip-only** **Example:** `Device(config-if)# tunnel mpls-ip-only`	Copies the Do Not Fragment bit from the inner IP header to the IP header of the tunnel packet. If the Do Not Fragment bit is not set, and the IP packet exceeds the MTU set for the interface, the payload is fragmented. When **tunnel path-mtu-discovery** is enabled, the **tunnel path-mtu-discovery** automatically gets enabled due to the dependency.
Step 12	**ip tcp mss** *mss-value* **Example:** `Device(config-if)# ip tcp mss 250`	(Optional) Specifies the maximum segment size (MSS) for TCP connections that originate or terminate on a router. • Use the *mss-value* argument to specify the maximum segment size for TCP connections, in bytes.
Step 13	**tunnel path-mtu-discovery** [**age-timer** {*aging-mins* | **infinite**}] **Example:** `Device(config-if)# tunnel path-mtu-discovery`	(Optional) Enables PMTUD on a GRE or IP-in-IP tunnel interface. • When PMTUD is enabled on a tunnel interface, PMTUD will operate for GRE IP tunnel packets to minimize fragmentation in the path between the tunnel endpoints.
Step 14	**end** **Example:** `Device(config-if)# end`	Exits interface configuration mode and returns to privileged EXEC mode.

What to Do Next

Proceed to the “Verifying Tunnel Configuration and Operation” section.

Configuring 6to4 Tunnels

Before you begin

With 6to4 tunnels, the tunnel destination is determined by the border-router IPv4 address, which is concatenated to the prefix 2002::/16 in the format 2002:*border-router-IPv4-address* ::/48. The border router at each end of a 6to4 tunnel must support both the IPv4 and IPv6 protocol stacks.

Note The configuration of only one IPv4-compatible tunnel and one 6to4 IPv6 tunnel is supported on a router. If you choose to configure both of these tunnel types on the same router, Cisco recommends that they not share the same tunnel source.

A 6to4 tunnel and an IPv4-compatible tunnel cannot share the same interface because both of them are NBMA "point-to-multipoint" access links, and only the tunnel source can be used to reorder the packets from a multiplexed packet stream into a single packet stream for an incoming interface. When a packet with an IPv4 protocol type of 41 arrives on an interface, the packet is mapped to an IPv6 tunnel interface on the basis of the IPv4 address. However, if both the 6to4 tunnel and the IPv4-compatible tunnel share the same source interface, the router cannot determine the IPv6 tunnel interface to which it should assign the incoming packet.

Manually configured IPv6 tunnels can share the same source interface because a manual tunnel is a "point-to-point" link, and both IPv4 source and the IPv4 destination of the tunnel are defined.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **interface tunnel** *tunnel-number*
4. **ipv6 address** *ipv6-prefix*/*prefix-length* [**eui-64**]
5. **tunnel source** {*ip-address* | *interface-type interface-number*}
6. **tunnel mode ipv6ip 6to4**
7. **exit**
8. **ipv6 route** *ipv6-prefix* / *prefix-length* **tunnel** *tunnel-number*
9. **end**

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**interface tunnel** *tunnel-number* **Example:** `Router(config)# interface tunnel 0`	Specifies a tunnel interface and number and enters interface configuration mode.
Step 4	**ipv6 address** *ipv6-prefix*/*prefix-length* [**eui-64**] **Example:**	Specifies the IPv6 address assigned to the interface and enables IPv6 processing on the interface.

	Command or Action	Purpose
	`Router(config-if)# ipv6 address 2002:c0a8:6301:1::1/64`	• The 32 bits following the initial 2002::/16 prefix correspond to an IPv4 address assigned to the tunnel source. **Note** See the "Configuring Basic Connectivity for IPv6" module for more information on configuring IPv6 addresses.
Step 5	**tunnel source** {*ip-address* | *interface-type interface-number*} **Example:** `Router(config-if)# tunnel source GigabitEthernet 0/0/0`	Specifies the source IPv4 address or the source interface type and number for the tunnel interface. **Note** The interface type and number specified in the **tunnel source** command must be configured with an IPv4 address.
Step 6	**tunnel mode ipv6ip 6to4** **Example:** `Router(config-if)# tunnel mode ipv6ip 6to4`	Specifies an IPv6 overlay tunnel using a 6to4 address.
Step 7	**exit** **Example:** `Router(config-if)# exit`	Exits interface configuration mode and returns to global configuration mode.
Step 8	**ipv6 route** *ipv6-prefix* / *prefix-length* **tunnel** *tunnel-number* **Example:** `Router(config)# ipv6 route 2002::/16 tunnel 0`	Configures a static route to the specified tunnel interface. **Note** When configuring a 6to4 overlay tunnel, you must configure a static route for the IPv6 6to4 prefix 2002::/16 to the 6to4 tunnel interface. • The tunnel number specified in the **ipv6 route** command must be the same tunnel number specified in the **interface tunnel** command.
Step 9	**end** **Example:** `Router(config)# end`	Exits global configuration mode and returns to privileged EXEC mode.

What to Do Next

Proceed to the “Verifying Tunnel Configuration and Operation” section.

Verifying Tunnel Configuration and Operation

The **show** and **ping** commands in the steps below can be used in any sequence. The following commands can be used for GRE tunnels, IPv6 manually configured tunnels, and IPv6 over IPv4 GRE tunnels.

SUMMARY STEPS

1. **enable**
2. **show interfaces tunnel** *number* [**accounting**]
3. **ping** [*protocol*] *destination*
4. **show ip route** [*address* [*mask*]]
5. **ping** [*protocol*] *destination*

DETAILED STEPS

Step 1 enable

Enables privileged EXEC mode. Enter your password if prompted.

Example:

```
Device> enable
```

Step 2 show interfaces tunnel *number* [accounting]

Two routers are configured to be endpoints of a tunnel. Device A has Gigabit Ethernet interface 0/0/0 configured as the source for tunnel interface 0 with an IPv4 address of 10.0.0.1 and an IPv6 prefix of 2001:0DB8:1111:2222::1/64. Device B has Gigabit Ethernet interface 0/0/0 configured as the source for tunnel interface 1 with an IPv4 address of 10.0.0.2 and an IPv6 prefix of 2001:0DB8:1111:2222::2/64.

To verify that the tunnel source and destination addresses are configured, use the **show interfaces tunnel** command on Device A.

Example:

```
Device A# show interfaces tunnel 0

Tunnel0 is up, line protocol is up
  Hardware is Tunnel
  MTU 1514 bytes, BW 9 Kbit, DLY 500000 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
  Tunnel source 10.0.0.1 (GigabitEthernet0/0/0), destination 10.0.0.2, fastswitch TTL 255
  Tunnel protocol/transport GRE/IP, key disabled, sequencing disabled
  Tunnel TTL 255
  Checksumming of packets disabled,  fast tunneling enabled
  Last input 00:00:14, output 00:00:04, output hang never
  Last clearing of "show interface" counters never
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue :0/0 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     4 packets input, 352 bytes, 0 no buffer
     Received 0 broadcasts, 0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     8 packets output, 704 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 output buffer failures, 0 output buffers swapped out
```

Step 3 ping [*protocol*] *destination*

To check that the local endpoint is configured and working, use the **ping** command on Device A.

Example:

```
DeviceA# ping 2001:0DB8:1111:2222::2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 2001:0DB8:1111:2222::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/20/20 ms
```

Step 4 **show ip route** [*address* [*mask*]]

To check that a route exists to the remote endpoint address, use the **show ip route** command.

Example:

```
DeviceA# show ip route 10.0.0.2

Routing entry for 10.0.0.0/24
  Known via "connected", distance 0, metric 0 (connected, via interface)
  Routing Descriptor Blocks:
  * directly connected, via GigabitEthernet0/0/0
      Route metric is 0, traffic share count is 1
```

Step 5 **ping** [*protocol*] *destination*

To check that the remote endpoint address is reachable, use the **ping** command on Device A.

Note The remote endpoint address may not be reachable using the **ping** command because of filtering, but the tunnel traffic may still reach its destination.

Example:

```
DeviceA# ping 10.0.0.2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.0.0.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/21/28 ms
```

To check that the remote IPv6 tunnel endpoint is reachable, use the **ping** command again on Device A. The note regarding filtering earlier in step also applies to this example.

Example:

```
DeviceA# ping 2001:0DB8:1111:2222::2

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 1::2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/20/20 ms
```

These steps may be repeated at the other endpoint of the tunnel.

Configuration Examples for Implementing Tunnels

Example: Configuring a GRE IPv4 Tunnel

The following example shows a simple configuration of GRE tunneling. Note that Gigabit Ethernet interface 0/0/1 is the tunnel source for Router A and the tunnel destination for Router B. Fast Ethernet interface 0/0/1 is the tunnel source for Router B and the tunnel destination for Router A.

Router A

```
interface Tunnel 0
 ip address 10.1.1.2 255.255.255.0
 tunnel source GigabitEthernet 0/0/1
 tunnel destination 192.168.3.2
 tunnel mode gre ip
!
interface GigabitEthernet 0/0/1
 ip address 192.168.4.2 255.255.255.0
```

Router B

```
interface Tunnel 0
 ip address 10.1.1.1 255.255.255.0
 tunnel source FastEthernet 0/0/1
 tunnel destination 192.168.4.2
 tunnel mode gre ip
!
interface FastEthernet 0/0/1
 ip address 192.168.3.2 255.255.255.0
```

The following example configures a GRE tunnel running both IS-IS and IPv6 traffic between Router A and Router B:

Router A

```
ipv6 unicast-routing
clns routing
!
interface Tunnel 0
 no ip address
 ipv6 address 2001:0DB8:1111:2222::1/64
 ipv6 router isis
 tunnel source GigabitEthernet 0/0/0
 tunnel destination 10.0.0.2
 tunnel mode gre ip
!
interface GigabitEthernet 0/0/0
 ip address 10.0.0.1 255.255.255.0
!
router isis
 network 49.0000.0000.000a.00
```

Router B

```
ipv6 unicast-routing
clns routing
!
interface Tunnel 0
 no ip address
 ipv6 address 2001:0DB8:1111:2222::2/64
 ipv6 router isis
 tunnel source GigabitEthernet 0/0/0
 tunnel destination 10.0.0.1
 tunnel mode gre ip
!
interface GigabitEthernet 0/0/0
 ip address 10.0.0.2 255.255.255.0
!
router isis
 network 49.0000.0000.000b.00
 address-family ipv6
 redistribute static
 exit-address-family
```

Example: Configuring EoMPLS over GRE

Router A Configuration

```
vrf definition VPN1
 rd 100:1
 address-family ipv4
 route-target both 100:1
 exit-address-family
 !
mpls label protocol ldp
mpls ldp neighbor 209.165.200.224 targeted
mpls ldp router-id Loopback0 force
!
interface tunnel 0
 ip address 209.165.200.225 255.255.255.224
 mpls label protocol ldp
 mpls ip
 keepalive 10 3
 tunnel source TenGigabitEthernet 2/1/0
 tunnel destination 209.165.200.226
!
interface Loopback 0
 ip address 209.165.200.230 255.255.255.224
 !
interface TenGigabitEthernet 2/1/0
 mtu 9216
 ip address 209.165.200.235 255.255.255.224
!
interface TenGigabitEthernet 9/1
 no ip address
!
interface TenGigabitEthernet 9/1.11
 vrf forwarding VPN1
 encapsulation dot1Q 300
 ip address 209.165.200.237 255.255.255.224
!
interface TenGigabitEthernet 9/2
```

```
 mtu 9216
 no ip address
xconnect 209.165.200.239 200 encapsulation mpls
!
router bgp 65000
 bgp log-neighbor-changes
 neighbor 209.165.200.240 remote-as 65000
 neighbor 209.165.200.240 update-source Loopback0
neighbor 209.165.200.245 remote-as 100
 !
address-family vpnv4
  neighbor 209.165.200.240 activate
  neighbor 209.165.200.240 send-community extended
 !
 address-family ipv4 vrf VPN1
  no synchronization
  neighbor 209.165.200.247 remote-as 100
  neighbor 209.165.200.248 activate
  neighbor 209.165.200.249 send-community extended
!
ip route 209.165.200.251 255.255.255.224 tunnel 0
ip route 209.165.200.254 255.255.255.224 209.165.200.256
Router B Configuration
vrf definition VPN1
 rd 100:1
 address-family ipv4
 route-target both 100:1
exit-address-family
 !
mpls ldp neighbor 209.165.200.229 targeted
mpls label protocol ldp
mpls ldp router-id Loopback0 force
!
interface tunnel 0
 ip address 209.165.200.230 255.255.255.224
 mpls label protocol ldp
 mpls ip
 keepalive 10 3
 tunnel source TenGigabitEthernet 3/3/0
 tunnel destination 209.165.200.232
!
interface Loopback 0
 ip address 209.165.200.234 255.255.255.224
!
interface TenGigabitEthernet 2/1/1
 mtu 9216
 no ip address
 xconnect 209.165.200.237 200 encapsulation mpls
!
interface TenGigabitEthernet 2/3/1
 mtu 9216
 no ip address
!
interface TenGigabitEthernet 2/3.11/1
 vrf forwarding VPN1
 encapsulation dot1Q 300
 ip address 209.165.200.239 255.255.255.224
!
interface TenGigabitEthernet 3/3/0
 mtu 9216
 ip address 209.165.200.240 255.255.255.224
!
router bgp 65000
 bgp log-neighbor-changes
```

```
 neighbor 209.165.200.241 remote-as 65000
 neighbor 209.165.200.241 update-source Loopback0
 neighbor 209.165.200.244 remote-as 200
 !
 address-family vpnv4
  neighbor 209.165.200.241 activate
  neighbor 209.165.200.241 send-community extended
 exit-address-family
 !
 address-family ipv4 vrf VPN1
  no synchronization
  neighbor 209.165.200.246 remote-as 200
  neighbor 209.165.200.246 activate
  neighbor 209.165.200.246 send-community extended
 exit-address-family
¡
ip route 209.165.200.226 255.255.255.224 tunnel 0
ip route 209.165.200.229 255.255.255.224 209.165.200.235
```

Configuring QoS Options on Tunnel Interfaces Examples

The following sample configuration applies GTS directly on the tunnel interface. In this example, the configuration shapes the tunnel interface to an overall output rate of 500 kb/s.

```
interface Tunnel 0
 ip address 10.1.2.1 255.255.255.0
 traffic-shape rate 500000 125000 125000 1000
 tunnel source 10.1.1.1
 tunnel destination 10.2.2.2
```

The following sample configuration shows how to apply the same shaping policy to the tunnel interface with the MQC commands:

```
policy-map tunnel
 class class-default
 shape average 500000 125000 125000
!
interface Tunnel 0
 ip address 10.1.2.1 255.255.255.0
 service-policy output tunnel
 tunnel source 10.1.35.1
 tunnel destination 10.1.35.2
```

Configuring QoS Options on Tunnel Interfaces Examples

The following sample configuration applies GTS directly on the tunnel interface. In this example, the configuration shapes the tunnel interface to an overall output rate of 500 kb/s.

```
interface Tunnel 0
 ip address 10.1.2.1 255.255.255.0
 traffic-shape rate 500000 125000 125000 1000
 tunnel source 10.1.1.1
 tunnel destination 10.2.2.2
```

The following sample configuration shows how to apply the same shaping policy to the tunnel interface with the MQC commands:

```
policy-map tunnel
 class class-default
 shape average 500000 125000 125000
!
interface Tunnel 0
 ip address 10.1.2.1 255.255.255.0
 service-policy output tunnel
 tunnel source 10.1.35.1
 tunnel destination 10.1.35.2
```

Policing Example

When an interface becomes congested and packets start to queue, you can apply a queueing method to packets that are waiting to be transmitted. Logical interfaces--tunnel interfaces in this example--do not inherently support a state of congestion and do not support the direct application of a service policy that applies a queueing method. Instead, you must apply a hierarchical policy. Create a "child" or lower-level policy that configures a queueing mechanism, such as low-latency queueing, with the **priority** command and CBWFQ with the **bandwidth** command.

```
policy-map child
 class voice
 priority 512
```

Create a "parent" or top-level policy that applies class-based shaping. Apply the child policy as a command under the parent policy because admission control for the child class is done according to the shaping rate for the parent class.

```
policy-map tunnel
 class class-default
 shape average 2000000
 service-policy child
```

Apply the parent policy to the tunnel interface.

```
interface tunnel 0
 service-policy tunnel
```

In the following example, a tunnel interface is configured with a service policy that applies queueing without shaping. A log message is displayed noting that this configuration is not supported.

```
Router(config)# interface tunnel1
Router(config-if)# service-policy output child
Class Based Weighted Fair Queueing not supported on this interface
```

Additional References

The following sections provide references related to implementing tunnels.

Related Documents

Related Topic	Document Title
All Cisco IOS XE commands	Cisco IOS Master Command List, All Releases .

Related Topic	Document Title
Tunnel commands: complete command syntax, command mode, defaults, command history, usage guidelines, and examples	*Cisco IOS Interface and Hardware Component Command Reference*
IPv6 commands: complete command syntax, command mode, defaults, command history, usage guidelines, and examples	*Cisco IOS IPv6 Command Reference*
Cisco IOS XE Interface and Hardware Component configuration modules	*Cisco IOS XE Interface and Hardware Component Configuration Guide*,
Cisco IOS XE IPv6 configuration modules	*Cisco IOS XE IPv6 Configuration Guide*,
Cisco IOS XE Quality of Service Solutions configuration modules	*Cisco IOS XE Quality of Service Solutions Configuration Guide*
Cisco IOS XE Multiprotocol Label Switching configuration modules	*Cisco IOS XE Multiprotocol Label Switching Configuration Guide*
Configuration example for a VRF-aware dynamic multipoint VPN (DMVPN)	"Dynamic Multipoint VPN (DMVPN)" configuration module in the *Cisco IOS XE Security Configuration Guide: Secure Connectivity*

Standards/RFCs

Standard	Title
No new or modified standards are supported, and support for existing standards has not been modified.	--
RFC 791	*Internet Protocol*
RFC 1191	*Path MTU Discovery*
RFC 1323	*TCP Extensions for High Performance*
RFC 1483	*Multiprotocol Encapsulation over ATM Adaptation Layer 5*
RFC 2003	*IP Encapsulation Within IP*
RFC 2018	*TCP Selective Acknowledgment Options*
RFC 2460	*Internet Protocol, Version 6 (IPv6)*
RFC 2473	*Generic Packet Tunneling in IPv6 Specification*
RFC 2474	*Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*
RFC 2516	*A Method for Transmitting PPP over Ethernet (PPPoE)*

Standard	Title
RFC 2547	*BGP/MPLS VPNs*
RFC 2780	*IANA Allocation Guidelines for Values in the Internet Protocol and Related Headers*
RFC 2784	*Generic Routing Encapsulation (GRE)*
RFC 2890	*Key and Sequence Number Extensions to GRE*
RFC 2893	*Transition Mechanisms for IPv6 Hosts and Routers*
RFC 3056	*Connection of IPv6 Domains via IPv4 Clouds*
RFC 3147	*Generic Routing Encapsulation over CLNS Networks*

Technical Assistance

Description	Link
The Cisco Support and Documentation website provides online resources to download documentation, software, and tools. Use these resources to install and configure the software and to troubleshoot and resolve technical issues with Cisco products and technologies. Access to most tools on the Cisco Support and Documentation website requires a Cisco.com user ID and password.	http://www.cisco.com/cisco/web/support/index.html

Feature Information for Implementing Tunnels

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 2: Feature Information for Implementing Tunnels

Feature Name	Releases	Feature Information
EoMPLS over GRE	Cisco IOS XE Release 2.5	The EoMPLS over GRE feature allows you to tunnel Layer 2 traffic through a Layer 3 MPLS network. This feature also helps to create the GRE tunnel as hardware-based switched, and with high performance that encapsulates EoMPLS frames within the GRE tunnel. No new commands were introduced or modified by this feature.

Feature Name	Releases	Feature Information
GRE Tunnel IP Source and Destination VRF Membership	Cisco IOS XE Release 2.2	The GRE Tunnel IP Source and Destination VRF Membership feature allows you to configure the source and destination of a tunnel to belong to any VPN VRF table. The following command was introduced or modified: **tunnel vrf**.
GRE Tunnel Keepalive	Cisco IOS XE Release 2.1	The GRE Tunnel Keepalive feature provides the capability of configuring keepalive packets to be sent over IP-encapsulated GRE tunnels. You can specify the rate at which keepalives will be sent and the number of times that a device will continue to send keepalive packets without a response before the interface becomes inactive. GRE keepalive packets may be sent from both sides of a tunnel or from just one side. The following command was introduced by this feature: **keepalive (tunnel interfaces)** .
IP over IPv6 Tunnels	Cisco IOS XE Release 2.4	The following commands were modified by this feature: **tunnel destination**, **tunnel mode**, and **tunnel source**.
IP Precedence for GRE Tunnels	Cisco IOS XE Release 2.1	This feature was introduced on Cisco ASR 1000 Aggregation Services Routers.
IP Tunnel— SSO	Cisco IOS XE Release 3.6	High availability support was added to IP Tunnels. No new commands were introduced or modified by this feature.
Tunnel ToS	Cisco IOS XE Release 2.1	The Tunnel ToS feature allows you to configure the ToS and Time-to-Live (TTL) byte values in the encapsulating IP header of tunnel packets for an IP tunnel interface on a router. The Tunnel ToS feature is supported in Cisco Express Forwarding, fast switching, and process switching forwarding modes. The following commands were introduced or modified by this feature: **show interfaces tunnel**, **tunnel tos**, **tunnel,** and **ttl**.