Tunnels Configuration Guide, 17.2.0

November 2017

Supporting AT&T Vyatta Network Operating System

Contents

Copyright Statement

About This Guide

This guide describes how to configure tunneling on AT&T products that run on the AT&T Vyatta Network Operating System (referred to as a virtual router, vRouter, or router in the guide).

Tunnels Overview

Overview

An IP tunneling protocol is a mechanism for encapsulating a packet from one network protocol into a packet from another protocol, thereby creating a “tunnel.” The transported protocol (the “passenger” protocol) is encapsulated by wrapping around it packet information for the tunneling protocol (the “carrier” protocol). The encapsulated packet is then forwarded to some destination and stripped of the encapsulating information, and the original packet is delivered.

The AT&T Vyatta vRouter supports three commonly used tunneling protocols.

- Generic Routing Encapsulation (GRE) tunnels can be used to carry non-IP protocols, such as Novell IPX, Banyan VINES, AppleTalk, and DECNet. They can also be used to carry multicast, broadcast, and IPv6 traffic.
- IP-in-IP tunnels can be used to carry only IPv4 traffic.
- Simple Internet Transition (SIT) tunnels can be used to transport IPv6 packets over IPv4 routing infrastructures.

A logical interface that sends IP packets in a tunneled mode is called a tunnel interface. A tunnel interface behaves just like any other system interface: you can configure routing protocols, firewall, NAT, and other features on them, and you can manage them by using standard operational tools and commands.

Note that GRE, IP-in-IP, and SIT tunnels are unencrypted.

GRE

This section presents the following topics:

The Generic Routing Encapsulation (GRE) protocol provides a simple general-purpose mechanism for encapsulating a packet from a wide variety of network protocols to be forwarded over another protocol. The original packet (the “passenger” packet) can be one of many network protocols—for example, a multicast packet, an IPv6 packet, or a non-IP LAN protocol, such as AppleTalk, Banyan VINES, or Novell IPX. The delivery protocol can be one of a number of routable IP protocols.

A GRE tunnel is stateless, which means that the protocol does not automatically monitor the state or availability of other endpoints. You can, however, direct the router to monitor the far end of the tunnel by sending keep-alive messages. If the other end of the tunnel becomes unavailable, its failure to respond to the messages alerts the router.

GRE uses the IP protocol number 47.

Applications for GRE

You might use GRE in the following situations:

- Connect networks that are running non-IP protocols, such as native LAN protocols, across the public IP network. Non-IP protocols, such as Novell IPX or Appletalk, are not routable across an IP network. A GRE tunnel allows you to create a virtual point-to-point link between two such networks over the public WAN.
- Route IPv6 packets across an IPv4 network, or connect any two similar networks across an infrastructure that uses different IP addressing.

- Encrypt multicast traffic. IPsec, which is a standard mechanism for providing security on IP networks, cannot encrypt multicast packets. However, multicast packets can be encapsulated within a GRE tunnel and then routed over a VPN connection, so that the encapsulated packets are protected by the IPsec tunnel.

Bridging with GRE

A limitation of a regular GRE-encapsulated tunnel is that the resulting tunnels cannot be added to a bridge group. GRE for bridging provides this ability. Use GRE for bridging only in cases in which tunnel interfaces are to be included in a bridge group.

To configure GRE for bridging, use the **`gre-bridge`** option of the **interfaces tunnel encapsulation** command. For more information about using bridging, refer to AT&T Vyatta Network Operating System Bridging Configuration Guide.

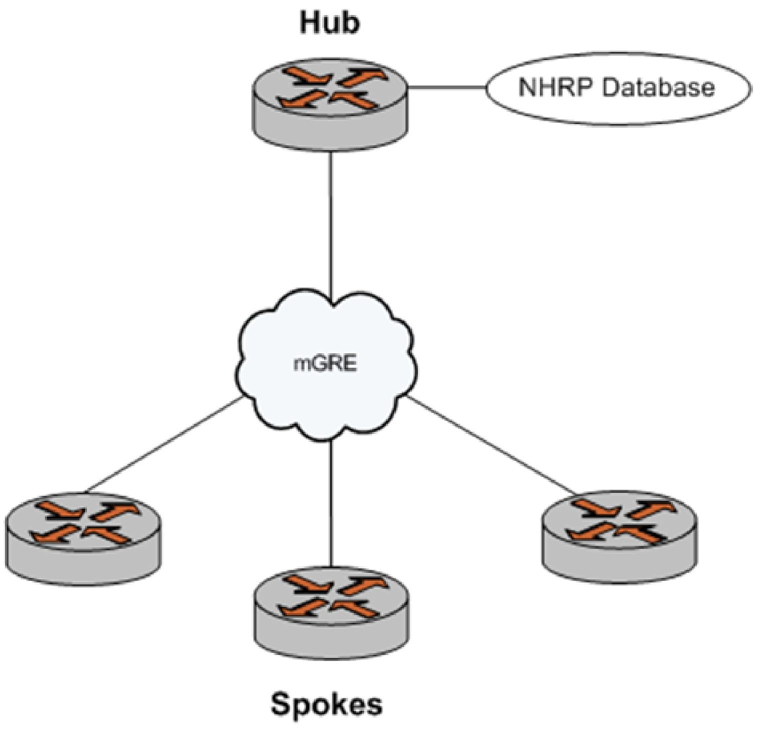
Multipoint GRE

A GRE tunnel, in its basic form, is essentially point to point. Supporting complex network topologies (such as hub-and-spoke and spoke-to-spoke technologies) with point-to-point tunnels is operationally problematic, requiring a full mesh of tunnels. Such a mesh also consumes a great deal of IP address space, as each pair of tunnel endpoints consumes a subnet. Multipoint GRE (mGRE) allows multiple destinations (for example, multiple spoke sites) to be grouped into a single multipoint interface.

To build the direct tunnels, mGRE uses the Next Hop Resolution Protocol (NHRP) addressing service. The hub maintains an NHRP database and the spokes query the hub database to obtain the IP addresses of the logical tunnel endpoints.

Note: The IPv6 transport mode for GRE and mGRE is not supported on the AT&T Vyatta vRouter.

Figure 1: Multipoint GRE tunnel



To use multipoint GRE, create a tunnel interface and specify **`gre-multipoint`** as the encapsulation type (by using the `interfaces tunnel <tunx> encapsulation <action>` command). The other main difference between a standard GRE configuration and an mGRE configuration is that, in an mGRE configuration, you do not specify an IP address for the remote endpoint (that is, you do not set the **`remote-ip`** parameter). Other parameters are configured as for ordinary GRE.

Both multipoint GRE and NHRP are necessary components for dynamic multipoint VPN (DMVPN), which is typically secured with IP Security (IPsec). DMVPN is discussed in AT&T Vyatta Network Operating System Bridging Configuration Guide. NHRP is discussed in AT&T Vyatta Network Operating System Services Configuration Guide.

Supported standards for GRE

The AT&T implementation of GRE complies with the following standards:

- RFC 1702: Generic Routing Encapsulation over IPv4 Networks
- RFC 2784: Generic Routing Encapsulation

IP-in-IP

This section presents the following topics:

- Applications for IP-in-IP *(page 10)*
- Supported standards for IP-in-IP *(page 10)*

The IP-in-IP encapsulation protocol is used to tunnel between networks that have different capabilities or policies. For example, an IP-in-IP tunnel can be used to forward a multicast packet across a section of a network (such as an IPsec tunnel) that does not support multicast routing. An IP-in-IP tunnel can also be used to influence the routing of a packet, or to deliver a packet to a mobile device that uses Mobile IP.

In IP-in-IP encapsulation, a second IP header is inserted in front of the IP header of the original packet (the “passenger” packet). The new IP header has as source and destination addresses the addresses of the tunnel endpoints. The IP header of the payload packet identifies the original sender and receiver. When the encapsulated packet exits the tunnel, the outer IP header is stripped off, and the original IP packet is delivered to the final destination.

Applications for IP-in-IP

IP-in-IP encapsulation is simple and robust. It is useful for connecting IPv4 networks that otherwise would not be able to communicate; however, it has some limitations:

- IP-in-IP encapsulation does not support broadcast traffic.
- IP-in-IP encapsulation does not support IPv6 traffic.

For forwarding this kind of traffic, GRE may be more appropriate.

Supported standards for IP-in-IP

The AT&T implementation of IP-in-IP complies with the following standard:

- RFC 1853: IP in IP Tunneling

SIT

This section presents the following topics:

- Applications for SIT *(page 11)*
- Supported standards for SIT *(page 11)*

The Simple Internet Transition (SIT) protocol is a mechanism for tunneling an IPv6 packet over IPv4 routing infrastructures. A SIT packet employs a dual-IP layer of IPv4 and IPv6 in hosts and routers for direct interoperability with nodes that implement both protocols. The encapsulation includes two IPv6 addressing structures that embed IPv4 addresses within IPv6 addresses.

SIT includes an optional mechanism for translating headers of IPv4 packets into IPv6, and the headers of IPv6 packets into IPv4. This option allows nodes that implement only IPv6 to interoperate with nodes that implement only IPv4.

Applications for SIT

The SIT protocol provides mechanisms for transitioning networks from IPv4 to IPv6. The embedded IPv4 address structure of SIT eliminates the need for tunnel configuration in most cases.

Supported standards for SIT

The AT&T implementation of SIT complies with the following standard:

- RFC 4213: Basic Transition Mechanisms for IPv6 Hosts and Routers

Securing tunnels

GRE, IP-in-IP, and SIT tunnels are not encrypted; they use a simple password-like key that is exchanged in clear text in each packet. They are not suitable for a production network unless otherwise secured. All GRE, IP-in-IP, and SIT tunnels can be protected by an IPsec tunnel. IPsec is explained in detail in AT&T Vyatta Network Operating System IPsec Site-to-Site VPN Configuration Guide.

Multipoint GRE (mGRE) tunnels can also be secured by using IPsec as part of a Dynamic Multipoint Virtual Private Network (DMVPN). Refer to AT&T Vyatta Network Operating System DMVPN Configuration Guide for more information on securing mGRE tunnels.

For information on determining which VPN solution best meets your needs, refer to AT&T Vyatta Network Operating System VPN Support Configuration Guide.

DHCP enabled interfaces as local tunnel endpoints

For NHRP spoke routes, you can configure a local tunnel endpoint by assigning a DHCP or local IP address. If DHCP is used, the hub end of the NHRP tunnel must be able to reach the DHCP subnet from which the spoke is assigned a local DHCP address. The encapsulation type must be GRE multipoint.

The local physical interface option is supported only for spoke routers, not hub routers. NHRP hub routes must be configured using a local IP address.

Using tunnels to extend IPsec capability

An IPsec policy-based tunnel cannot directly route non-IP or multicast protocols. IPsec also has limitations from an operations point of view.

Using tunnel interfaces with IPsec VPN provides secure, routable tunnel connections between gateways. These tunnels have some advantages over traditional IPsec policy-based tunnel mode connections.

- They support standard operational commands, such as `show interfaces` and `show route`.
- They support operational tools, such as traceroute and SNMP.
- They provide dynamic tunnel failover by using routing protocols.
- They simplify IPsec policies and troubleshooting.

IPsec is explained in detail in AT&T Vyatta Network Operating System IPsec Site-to-Site VPN Configuration Guide. See that guide for more information.

The use of tunnel interfaces with IPsec is documented in the following standard, which describes the use of IP-in-IP tunnels that is combined with IPsec transport mode encryption to provide secure routable tunnels:

- RFC 3884: Use of IPsec Transport Mode for Dynamic Routing

Another method of providing a secure routable interface is to use a Virtual Tunnel Interface (VTI). Refer to AT&T Vyatta Network Operating System IPsec Site-to-Site VPN Configuration Guide for more information.

Tunnel Configuration Examples

Before you begin

The following examples have some common elements.

- Any Ethernet (data plane) or loopback interface to be used must already be configured. The examples do not show these configurations.
- The examples show both Ethernet and loopback interfaces being configured as tunnel endpoints. Configuring a loopback interface as the tunnel endpoint is advantageous in systems in which multiple paths between tunnel endpoints exist because the tunnel does not fail if an Ethernet interface fails.

Refer to AT&T Vyatta Network Operating System LAN Interfaces Configuration Guide for information on configuring Ethernet and loopback interfaces.

Note: In the AT&T Vyatta vRouter, a data plane interface is an abstraction that represents the underlying physical or virtual Ethernet interface of the system. The terms Ethernet interface and data plane interface are synonymous in this guide.

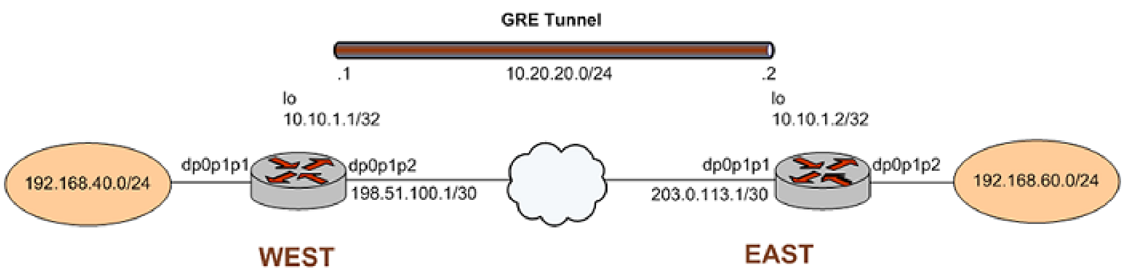
GRE tunnel connecting remote networks

This section presents a sample configuration for a basic GRE tunnel between the WEST and EAST AT&T Vyatta vRouters. First, WEST is configured, and then EAST.

This basic tunnel is not protected by a key, which means that it is not secure and would not be suitable for a production network unless otherwise secured.

When you finish the steps, these systems are configured as shown in the following figure. In the figure, note that the remote endpoint must be reachable.

Figure 2: Basic GRE tunnel



Configure WEST

The GRE tunnel in the sample configuration extends from WEST through the wide-area network to EAST. In this example, you create the tunnel interface and the tunnel endpoint on WEST.

- The tun0 tunnel interface on WEST is assigned the 10.20.20.1 IP address on 10.20.20.0/24 network.
- The source IP address of the tunnel endpoint (local-ip) is the same as the address associated with the loopback interface (lo) in this example.
- The IP address of the other end of the tunnel (remote-ip) is the address of the loopback interface on EAST.
- A static route is created to specify how to get to the remote LAN through the tunnel.

The following table shows how to create the tunnel interface and the tunnel endpoint on WEST. To do this, perform the following steps on WEST in configuration mode.

Table 1: Creating a basic GRE tunnel endpoint on WEST

Step	Command

Create the tunnel interface, and specify the IP address to be associated with it.	`vyatta@WEST# set interfaces tunnel tun0`
	` address 10.20.20.1/24`

Specify the source IP address for the tunnel.	`vyatta@WEST# set interfaces tunnel tun0`
	` local-ip 10.10.1.1`

Specify the IP address of the other end of the tunnel.	`vyatta@WEST# set interfaces tunnel tun0`
	` remote-ip 10.10.1.2`

Specify the encapsulation mode for the tunnel.	`vyatta@WEST# set interfaces tunnel tun0`
	` encapsulation gre`

Assign a brief description to the tunnel interface.	`vyatta@WEST# set interfaces tunnel tun0`
	` description “GRE tunnel to EAST”`

Commit the configuration.	`vyatta@WEST# commit`

View the configuration.	`vyatta@WEST# show interfaces tunnel tun0`
	` address 10.20.20.1/24`
	` description “Tunnel to EAST”`
	` encapsulation gre`
	` local-ip 10.10.1.1`
	` remote-ip 10.10.1.2`

Create a static route to access the remote subnet through the tunnel.	`vyatta@WEST# set protocols static route`
	` 192.168.60.0/24 next-hop 10.20.20.2`

Commit the configuration.	`vyatta@WEST# commit`

View the configuration.	`vyatta@WEST# show protocols`
	` static {`
	`  route 192.168.60.0/24 {`
	`   next-hop 10.20.20.2 {`
	`   }`
	`  }`
	` }`

Configure EAST

In this example, you create the tunnel endpoint on EAST.

- The tun0 tunnel interface on EAST is assigned the 10.20.20.2 IP address on 10.20.20.0/24 network.
- The source IP address of the tunnel endpoint (local-ip) is the same as the address associated with the loopback interface (lo) in this example.
- The IP address of the other end of the tunnel (remote-ip) is the address of the loopback interface on WEST.
- A static route is created to specify how to get to the remote LAN through the tunnel.

The following table shows how to create the tunnel endpoint on EAST. To do this, perform the following steps on EAST in configuration mode.

Table 2: Create a basic tunnel endpoint on EAST

Step	Command

Create the tunnel interface, and specify the IP address to be associated with it.	`vyatta@EAST# set interfaces tunnel tun0`
	` address 10.20.20.2/24`

Specify the source IP address for the tunnel.	`vyatta@EAST# set interfaces tunnel tun0`
	` local-ip 10.10.1.2`

Specify the IP address of the other end of the tunnel.	`vyatta@EAST# set interfaces tunnel tun0`
	` remote-ip 10.10.1.1`

Specify the encapsulation mode for the tunnel.	`vyatta@EAST# set interfaces tunnel tun0`
	` encapsulation gre`

Assign a brief description to the tunnel interface.	`vyatta@EAST# set interfaces tunnel tun0`
	` description “GRE tunnel to WEST”`

Commit the configuration.	`vyatta@EAST# commit`

View the configuration.	`vyatta@EAST# show interfaces tunnel tun0`
	` address 10.20.20.2/24`
	` description “Tunnel to WEST”`
	` encapsulation gre`
	` local-ip 10.10.1.2`
	` remote-ip 10.10.1.1`

Create a static route to access the remote subnet through the tunnel.	`vyatta@EAST# set protocols static route`
	` 192.168.40.0/24 next-hop 10.20.20.1`

Commit the configuration.	`vyatta@EAST# commit`

Step	Command

View the configuration.	`vyatta@EAST# show protocols`
	` static {`
	`  route 192.168.40.0/24 {`
	`   next-hop 10.20.20.1 {`
	`   }`
	`  }`
	` }`

A GRE tunnel with authentication

In this section, some additional parameters are specified for the tunnel interfaces that are defined in the previous section.

- A key is specified so that the hosts can authenticate each other. This key must match on the two endpoints. Note that authentication is not encryption.
- The time to live (TTL), Type of Service (ToS), and maximum transmission unit (MTU) are specified for each endpoint.
- A firewall rule set is applied to each tunnel interface.

Configure WEST

The following table shows how to specify additional values for the tunnel endpoint on WEST that is created in Configure WEST *(page 12)*.

- A key, 101088, is provided as a password-like mechanism. The key on each side must match.
- The TTL for packets is set to 220, ToS field is set to 55, and MTU for packets is set to 1460.
- Two firewall rule sets are applied to the tunnel interface:
  - The tun0-fw-in rule set is applied to packets ingressing through the tunnel interface.
  - The tun0-fw-out rule set is applied to packets egressing through the tunnel interface.

  In the example, it is assumed that these firewall rule sets have already been defined. For information on defining firewall rule sets, refer to AT&T Vyatta Network Operating System Firewall Configuration Guide.

To configure the GRE tunnel endpoint, perform the following steps on WEST in configuration mode.

Table 3: Adding values to the GRE tunnel endpoint on WEST

Step	Command

Provide the authentication key.	`vyatta@WEST# set interfaces tunnel tun0`
	` parameters ip key 101088`

Set the TTL.	`vyatta@WEST# set interfaces tunnel tun0`
	` parameters ip ttl 220`

Set the ToS.	`vyatta@WEST# set interfaces tunnel tun0`
	` parameters ip tos 55`

Step	Command
Set the MTU.	`vyatta@WEST# set interfaces tunnel tun0 mtu 1460`
Apply the firewall rule set for incoming packets.	`vyatta@WEST# set interfaces tunnel tun0 firewall in name tun0-fw-in`
Apply the firewall rule set for outgoing packets.	`vyatta@WEST# set interfaces tunnel tun0 firewall out name tun0-fw-out`
Commit the configuration.	`vyatta@WEST# commit`
View the configuration.	See below

```
vyatta@WEST# show interfaces tunnel tun0
 address 10.20.20.1/24
 description “Tunnel to EAST”
 encapsulation gre
 firewall
  in {
   name tun0-fw-in
  }
  out {
   name tun0-fw-out
  }
 }
 local-ip 10.10.1.1
 mtu 1460
 parameters {
 ip {
  key 101088
   tos 55
   ttl 220
 }
}
 remote-ip 10.10.1.2
```

Configure EAST

The following table shows how to specify additional values for the tunnel endpoint on EAST that is created in Configure EAST *(page 14)*.

- A key 101088 is provided as a password-like mechanism. This value matches the key configured for WEST.
- The TTL for packets is set to 220, ToS field is set to 55, and MTU for packets is set to 1460.
- Two firewall rule sets are applied to the tunnel interface:
  - The tun0-fw-in rule set is applied to packets ingressing through the tunnel interface.
  - The tun0-fw-out rule set is applied to packets egressing through the tunnel interface.

  In the example, it is assumed that these firewall rule sets have already been defined. For information on defining firewall rule sets, refer to AT&T Vyatta Network Operating System Firewall Configuration Guide.

To configure the GRE tunnel endpoint, perform the following steps on EAST in configuration mode.

Table 4: Adding values to the GRE tunnel endpoint on EAST

Step	Command

Provide the authentication key.	`vyatta@EAST# set interfaces tunnel tun0`
	` parameters ip key 101088`

Set the TTL.	`vyatta@EAST# set interfaces tunnel tun0`
	` parameters ip ttl 220`

Set the ToS.	`vyatta@EAST# set interfaces tunnel tun0`
	` parameters ip tos 55`

Set the MTU.	`vyatta@EAST# set interfaces tunnel tun0 mtu`
	` 1460`

Apply the firewall rule set for incoming packets.	`vyatta@EAST# set interfaces tunnel tun0`
	` firewall in name tun0-fw-in`

Apply the firewall rule set for outgoing packets.	`vyatta@EAST# set interfaces tunnel tun0`
	` firewall out name tun0-fw-out`

Commit the configuration.	`vyatta@EAST# commit`

View the configuration.	`vyatta@EAST# show interfaces tunnel tun0`
	` address 10.20.20.2/24`
	` description “Tunnel to WEST”`
	` encapsulation gre`
	` firewall`
	`  in {`
	`   name tun0-fw-in`
	`  }`
	`  out {`
	`   name tun0-fw-out`
	`  }`
	` }`
	` local-ip 10.10.1.2`
	` mtu 1460`
	` parameters {`
	` ip {`
	`  key 101088`
	`   tos 55`
	`   ttl 220`
	` }`
	`}`
	` remote-ip 10.10.1.1`

Multipoint GRE tunnels

This section presents a sample configuration for a basic multipoint GRE (mGRE) tunnel between the HUB and SPOKE1 AT&T Vyatta vRouter and one between HUB and SPOKE2.

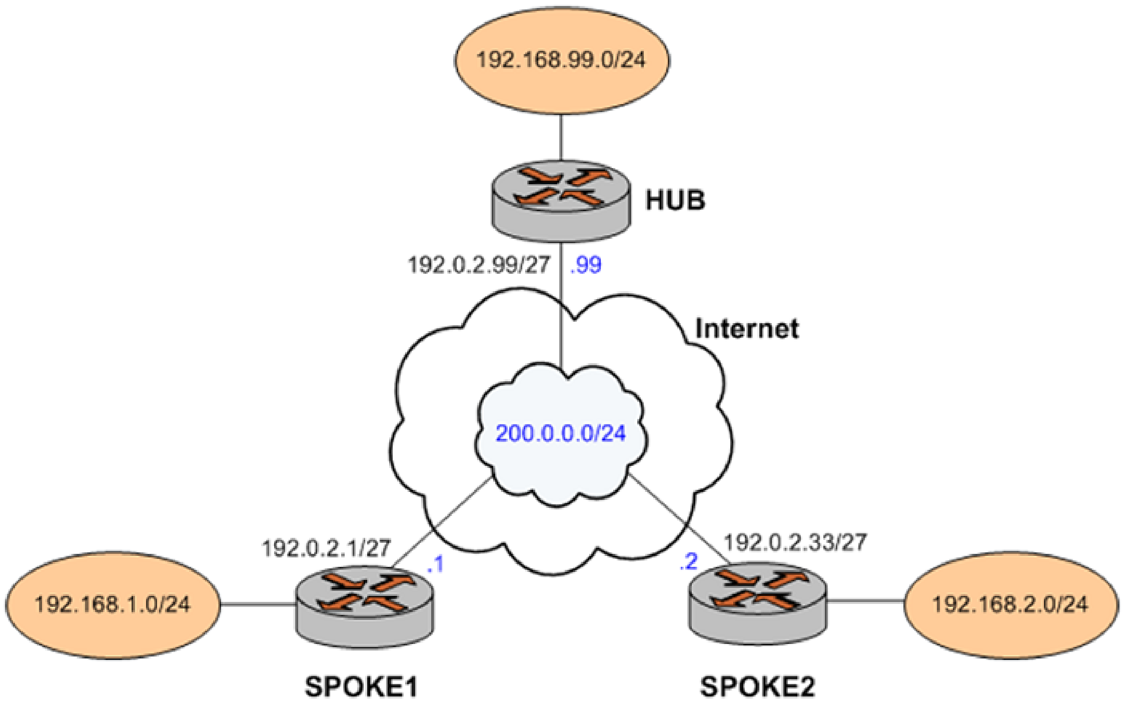
The configuration shown in this example also provides for a dynamic tunnel to be created between SPOKE1 and SPOKE2, as required. The ability to form a dynamic tunnel directly between the spokes derives from the use of mGRE and Next Hop Resolution Protocol (NHRP). This configuration can be expanded by creating additional spoke nodes with no change to the HUB configuration. For more information on NHRP, refer to AT&T Vyatta Network Operating System Services Configuration Guide.

Note that spoke-to-spoke traffic does not pass through the HUB. Note also that a typical production environment would use a routing protocol such as OSPF rather than static routes, which are used in this example.

The basic mGRE tunnels presented in this example are not protected by IPsec encryption, which means they are not secure and would not be suitable for a production network unless otherwise secured. Dynamic multipoint VPN (DMVPN) uses mGRE, NHRP, and IPsec to provide a secure hub-and-spoke tunnel environment. For more information on creating a DMVPN environment, see AT&T Vyatta Network Operating System DMVPN Configuration Guide.

When this example is completed, the network will be configured as shown in the following figure.

Figure 3: Basic mGRE tunnel network



Configure HUB

Two multipoint GRE tunnels are configured. One is between HUB and SPOKE1. The other is between HUB and SPOKE2. The first step is to configure HUB.

In this example, you create the tunnel interface and the tunnel endpoint on HUB.

- The tunnel interface tun0 on HUB is assigned the IP address 200.0.0.99 on subnet 200.0.0.0/24.
- The source IP address of the tunnel endpoint (the `local-ip`) is the same as the address associated with the local Ethernet interface in this example (192.0.2.99/24).

- A static route is created to specify how to get to the remote LANs through the tunnel.

Table 5: Creating a multipoint GRE endpoint on HUB

Step	Command
Create the tunnel interface, and specify the IP address to be associated with it.	`vyatta@HUB# set interfaces tunnel tun0 address 200.0.0.99/24`
Specify the encapsulation mode for the tunnel.	`vyatta@HUB# set interfaces tunnel tun0 encapsulation gre-multipoint`
Specify the source IP address for the tunnel. This address is the IP address of the physical interface for the tunnel endpoint.	`vyatta@HUB# set interfaces tunnel tun0 local-ip 192.0.2.99`
Allow multicast protocols (for example, routing protocols) to be carried over the tunnel.	`vyatta@HUB# set interfaces tunnel tun0 multicast enable`
Specify an authentication key for the NHRP network.	`vyatta@HUB# set interfaces tunnel tun0 nhrp authentication pre-shared-secret NET123`
Specify the hold time for the NHRP network.	`vyatta@HUB# set interfaces tunnel tun0 nhrp holding-time 300`
Specify that multicast packets are to be forwarded to all directly connected peers.	`vyatta@HUB# set interfaces tunnel tun0 nhrp multicast parameters dynamic`
Specify that Cisco-style NHRP Traffic Indication packets are to be sent.	`vyatta@HUB# set interfaces tunnel tun0 nhrp redirect`
Specify an authentication key for the tunnel.	`vyatta@HUB# set interfaces tunnel tun0 parameters ip key 1`
Commit the configuration.	`vyatta@HUB# commit`

Step	Command

View the configuration.	`vyatta@HUB# show interfaces tunnel`
	` tun0 {`
	`     address 200.0.0.99/24`
	`     encapsulation gre-multipoint`
	`     local-ip 192.0.2.99`
	`     multicast enable`
	`     nhrp {`
	`         authentication NET123`
	`         holding-time 300`
	`         multicast {`
	`             parameters dynamic`
	`         }`
	`         redirect`
	`     }`
	`     parameters {`
	`         ip {`
	`             key 1`
	`         }`
	`     }`
	` }`

Create a static route to access the remote LAN behind SPOKE1 through the tunnel.	`vyatta@WEST# set protocols static route`
	` 192.168.1.0/24 next-hop 200.0.0.1`

Create a static route to access the remote LAN behind SPOKE2 through the tunnel.	`vyatta@WEST# set protocols static route`
	` 192.168.2.0/24 next-hop 200.0.0.2`

Commit the configuration.	`vyatta@WEST# commit`

View the configuration.	`vyatta@WEST# show protocols`
	` static {`
	`  route 192.168.1.0/24 {`
	`   next-hop 200.0.0.1 {`
	`   }`
	`  }`
	`  route 192.168.2.0/24 {`
	`   next-hop 200.0.0.2 {`
	`   }`
	`  }`
	` }`

Configure SPOKE1

The second step is to configure SPOKE1.

In this example, you create the tunnel interface and the tunnel endpoint on SPOKE1.

- The tunnel interface tun0 on HUB is assigned the IP address 200.0.0.1 on subnet 200.0.0.0/24.
- The source IP address of the tunnel endpoint (the `local-ip`) is the same as the address associated with the local Ethernet interface in this example (192.0.2.1/24).
- A static route is created to specify how to get to the remote LANs through the tunnel.

Table 6: Creating a multipoint GRE endpoint on SPOKE1

Step	Command
Create the tunnel interface, and specify the IP address to be associated with it.	`vyatta@SPOKE1# set interfaces tunnel tun0 address 200.0.0.1/24`
Specify the encapsulation mode for the tunnel.	`vyatta@SPOKE1# set interfaces tunnel tun0 encapsulation gre-multipoint`
Specify the source IP address for the tunnel. This address is the IP address of the physical interface for the tunnel endpoint.	`vyatta@SPOKE1# set interfaces tunnel tun0 local-ip 192.0.2.1`
Allow multicast protocols (for example, routing protocols) to be carried over the tunnel.	`vyatta@SPOKE1# set interfaces tunnel tun0 multicast enable`
Specify an authentication key for the NHRP network.	`vyatta@SPOKE1# set interfaces tunnel tun0 nhrp authentication pre-shared-secret NET123`
Map the IP address of the tunnel interface of the Hub to its physical IP address.	`vyatta@SPOKE1# set interfaces tunnel tun0 nhrp map 200.0.0.99/24 nbma-address 192.0.2.99`
Specify that this spoke should register itself automatically on startup.	`vyatta@SPOKE1# set interfaces tunnel tun0 nhrp map 200.0.0.99/24 register`
Specify that multicast packets are to be repeated to each statically configured next hop.	`vyatta@SPOKE1# set interfaces tunnel tun0 nhrp multicast parameters nhs`
Specify that Cisco-style NHRP Traffic Indication packets are to be sent.	`vyatta@SPOKE1# set interfaces tunnel tun0 nhrp redirect`
Specify that shortcut routes can be created.	`vyatta@SPOKE1# set interfaces tunnel tun0 nhrp shortcut`
Specify an authentication key for the tunnel.	`vyatta@SPOKE1# set interfaces tunnel tun0 parameters ip key 1`
Commit the configuration.	`vyatta@SPOKE1# commit`

Step	Command

View the configuration.	
	```
	vyatta@SPOKE1# show interfaces tunnel
	 tun0 {
	     address 200.0.0.1/24
	     encapsulation gre-multipoint
	     local-ip 192.0.2.1
	     multicast enable
	     nhrp {
	         authentication NET123
	         map 200.0.0.99/24 {
	             nbma-address 192.0.2.99
	             register
	         }
	         multicast {
	             parameters nhs
	         }
	         redirect
	         shortcut
	     }
	     parameters {
	         ip {
	             key 1
	         }
	     }
	 }
	```

Create a static route to access the remote LAN behind HUB through the tunnel.	
	```
	vyatta@WEST# set protocols static route
	 192.168.99.0/24 next-hop 200.0.0.99
	```

Create a static route to access the remote LAN behind SPOKE2 through the tunnel.	
	```
	vyatta@WEST# set protocols static route
	 192.168.2.0/24 next-hop 200.0.0.2
	```

Commit the configuration.	`vyatta@WEST# commit`

View the configuration.	
	```
	vyatta@WEST# show protocols
	 static {
	  route 192.168.99.0/24 {
	   next-hop 200.0.0.99 {
	   }
	  }
	  route 192.168.2.0/24 {
	   next-hop 200.0.0.2 {
	   }
	  }
	 }
	```

Configure SPOKE2

The final step is to configure SPOKE2.

In this example, you create the tunnel interface and the tunnel endpoint on SPOKE2.

- The tunnel interface tun0 on HUB is assigned the IP address 200.0.0.2 on subnet 200.0.0.0/24.

- The source IP address of the tunnel endpoint (the `local-ip`) is the same as the address associated with the local Ethernet interface in this example (192.0.2.33/24).
- A static route is created to specify how to get to the remote LANs through the tunnel.

Table 7: Creating a multipoint GRE endpoint on SPOKE2

Step	Command
Create the tunnel interface, and specify the IP address to be associated with it.	`vyatta@SPOKE2# set interfaces tunnel tun0 address 200.0.0.2/24`
Specify the encapsulation mode for the tunnel.	`vyatta@SPOKE2# set interfaces tunnel tun0 encapsulation gre-multipoint`
Specify the source IP address for the tunnel. This address is the IP address of the physical interface for the tunnel endpoint.	`vyatta@SPOKE2# set interfaces tunnel tun0 local-ip 192.0.2.33`
Allow multicast protocols (for example, routing protocols) to be carried over the tunnel.	`vyatta@SPOKE2# set interfaces tunnel tun0 multicast enable`
Specify an authentication key for the NHRP network.	`vyatta@SPOKE2# set interfaces tunnel tun0 nhrp authentication pre-shared-secret NET123`
Map the IP address of the tunnel interface of the Hub to its physical IP address.	`vyatta@SPOKE2# set interfaces tunnel tun0 nhrp map 200.0.0.99/24 nbma-address 192.0.2.99`
Specify that this spoke should register itself automatically on startup.	`vyatta@SPOKE2# set interfaces tunnel tun0 nhrp map 200.0.0.99/24 register`
Specify that multicast packets are to be repeated to each statically configured next hop.	`vyatta@SPOKE2# set interfaces tunnel tun0 nhrp multicast parameters nhs`
Specify that Cisco-style NHRP Traffic Indication packets are to be sent.	`vyatta@SPOKE2# set interfaces tunnel tun0 nhrp redirect`
Specify that shortcut routes can be created.	`vyatta@SPOKE2# set interfaces tunnel tun0 nhrp shortcut`
Specify an authentication key for the tunnel.	`vyatta@SPOKE2# set interfaces tunnel tun0 parameters ip key 1`

Step	Command
Commit the configuration.	`vyatta@SPOKE2# commit`

View the configuration.	
	```
	vyatta@SPOKE2# show interfaces tunnel
	 tun0 {
	     address 200.0.0.2/24
	     encapsulation gre-multipoint
	     local-ip 192.0.2.33
	     multicast enable
	     nhrp {
	         authentication NET123
	         map 200.0.0.99/24 {
	             nbma-address 192.0.2.99
	             register
	         }
	         multicast {
	             parameters nhs
	         }
	         redirect
	         shortcut
	     }
	     parameters {
	         ip {
	             key 1
	         }
	     }
	 }
	```

Create a static route to access the remote LAN behind HUB through the tunnel.	
	```
	vyatta@WEST# set protocols static route
	 192.168.99.0/24 next-hop 200.0.0.99
	```

Create a static route to access the remote LAN behind SPOKE1 through the tunnel.	
	```
	vyatta@WEST# set protocols static route
	 192.168.1.0/24 next-hop 200.0.0.1
	```

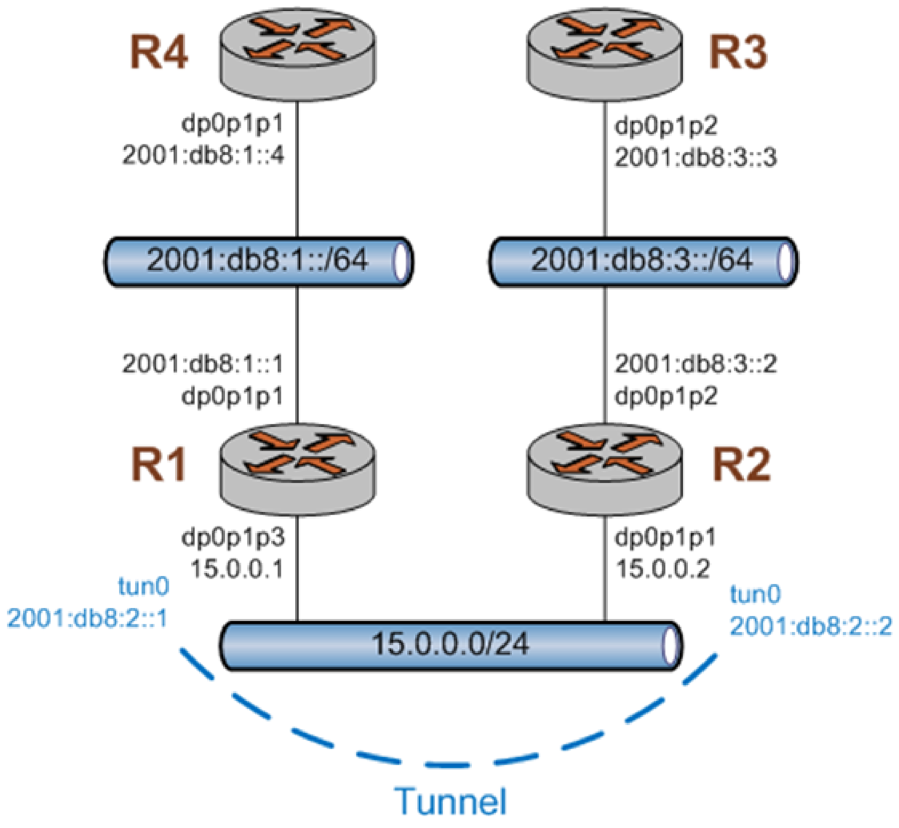
Commit the configuration.	`vyatta@WEST# commit`

View the configuration.	
	```
	vyatta@WEST# show protocols
	 static {
	  route 192.168.99.0/24 {
	   next-hop 200.0.0.99 {
	   }
	  }
	  route 192.168.1.0/24 {
	   next-hop 200.0.0.1 {
	   }
	  }
	 }
	```

Tunneling IPv6 traffic in IPv4 with SIT

Figure 1 *(page 25)* shows a network with four nodes. Each of R1 and R2 has an interface that is configured as IPv6 and an interface that is configured as IPv4. The figure shows configuration of the nodes by using tunneling over IPv4 to enable R3 and R4 to communicate through R1 and R2.

Figure 4: Tunneling IPv6 traffic in IPv4



In the figure, all interfaces are configured with the IP addresses as shown, and R1 and R2 have forwarding enabled.

Create a SIT tunnel

R1 and R2 must be configured to create a tunnel between them to encapsulate the IPv6 traffic. To configure R1 to create a tunnel that uses SIT encapsulation between 15.0.0.1 and 15.0.0.2, perform the following steps in configuration mode.

Table 8: Configuring a tunnel interface on R1

Step	Command
Create a tunnel with SIT encapsulation.	`vyatta@R1# set interfaces tunnel tun0 encapsulation sit`
Specify the local IP address.	`vyatta@R1# set interfaces tunnel tun0 local-ip 15.0.0.1`

Step	Command
Specify the remote IP address.	`vyatta@R1# set interfaces tunnel tun0 remote-ip 15.0.0.2`
Configure the IPv6 address on the interface.	`vyatta@R1# set interfaces tunnel tun0 address 2001:db8:2::1/64`
Commit the changes.	`vyatta@R1# commit`

To configure R2 to create a tunnel that uses SIT encapsulation between 15.0.0.2 and 15.0.0.1, perform the following steps in configuration mode.

Table 9: Configuring a tunnel interface on R2

Step	Command
Create a tunnel with SIT encapsulation.	`vyatta@R2# set interfaces tunnel tun0 encapsulation sit`
Specify the local IP address.	`vyatta@R2# set interfaces tunnel tun0 local-ip 15.0.0.2`
Specify the remote IP address.	`vyatta@R2# set interfaces tunnel tun0 remote-ip 15.0.0.1`
Configure the IPv6 address on the interface.	`vyatta@R2# set interfaces tunnel tun0 address 2001:db8:2::2/64`
Commit the changes.	`vyatta@R2# commit`

At this point, connectivity exists between R1 and R2 across the tunnel interface. The following example shows the capture of a ping from 2001:db8:2::1 to 2001:db8:2::2. Notice that the IPv6 ping packet is encapsulated by the IPv4 header.

Capture of a ping

```
Frame 22 (138 bytes on wire, 138 bytes captured)
Ethernet II, Src: Vmware_d6:81:80 (00:0c:29:d6:81:80), Dst: Vmware_4e:fc:b6 (00:0c:29:4e:fc:b6)
    Destination: Vmware_4e:fc:b6 (00:0c:29:4e:fc:b6)
    Source: Vmware_d6:81:80 (00:0c:29:d6:81:80)
    Type: IP (0x0800)
Internet Protocol, Src: 15.0.0.1 (15.0.0.1), Dst: 15.0.0.2 (15.0.0.2)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 124
    Identification: 0x0000 (0)
    Flags: 0x04 (Don't Fragment)
    Fragment offset: 0
    Time to live: 255
    Protocol: IPv6 (0x29)
    Header checksum: 0x5d56 [correct]
    Source: 15.0.0.1 (15.0.0.1)
    Destination: 15.0.0.2 (15.0.0.2)
Internet Protocol Version 6
    0110 .... = Version: 6
    .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 64
    Next header: ICMPv6 (0x3a)
    Hop limit: 64
    Source: 2001:db8:2::1 (2001:db8:2::1)
    Destination: 2001:db8:2::2 (2001:db8:2::2)
Internet Control Message Protocol v6
    Type: 129 (Echo reply)
    Code: 0
    Checksum: 0x2fca [correct]
    ID: 0xe825
    Sequence: 0x001b
    Data (56 bytes)
0000  9b a8 25 49 58 0c 07 00 08 09 0a 0b 0c 0d 0e 0f   ..%IX...........
0010  10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f   ................
0020  20 21 22 23 24 25 26 27 28 29 2a 2b 2c 2d 2e 2f    !"#$%&'()*+,-./
0030  30 31 32 33 34 35 36 37                           01234567
```

Tunnel Commands

Related tunnel commands

This chapter presents the commands for configuring GRE and IP-in-IP tunnels.

Commands for using other system features with tunnel interfaces are described in the following noted guides.

Related Commands Documented Elsewhere	
Bridging	Commands for configuring bridge groups on tunnel interfaces are described in AT&T Vyatta Network Operating System Bridging Configuration Guide.
Firewall	Commands for configuring firewall on tunnel interfaces are described in AT&T Vyatta Network Operating System Firewall Configuration Guide.
IGMP	Commands for configuring IGMP and MLD parameters on tunnel interfaces are described in AT&T Vyatta Network Operating System IGMP and MLD Configuration Guide.
Multicast	Commands for configuring multicast IP parameters on tunnel interfaces are described in AT&T Vyatta Network Operating System Multicast Routing Configuration Guide.
NHRP	Commands for configuring NHRP parameters on tunnel interfaces are described in AT&T Vyatta Network Operating System Services Configuration Guide.
OSPF	Commands for configuring OSPF parameters on tunnel interfaces are described in AT&T Vyatta Network Operating System OSPF Configuration Guide.
OSPFv3	Commands for configuring OSPFv3 parameters on tunnel interfaces are described in AT&T Vyatta Network Operating System OSPFv3 Configuration Guide.
PIM	Commands for configuring PIM parameters on tunnel interfaces are described in AT&T Vyatta Network Operating System PIM Configuration Guide.
Policy-based routing	Commands for configuring policy-based routing on tunnel interfaces are described in AT&T Vyatta Network Operating System Policy-based Routing Configuration Guide.
QoS	Commands for configuring quality of service on tunnel interfaces are described in AT&T Vyatta Network Operating System QoS Configuration Guide.
RIP	Commands for configuring RIP parameters on tunnel interfaces are described in AT&T Vyatta Network Operating System RIP Configuration Guide.

Related Commands Documented Elsewhere	
RIPng	Commands for configuring the RIPng on tunnel interfaces are described in AT&T Vyatta Network Operating System RIPng Configuration Guide.

clear interfaces tunnel counters

Clears statistics for a tunnel interface.

Syntax:
`clear interfaces tunnel` { **`counters`** | *tunx* **`counters`** }

tunx

Clears information for the specified tunnel interface. The range is tun0 through tun*x*, where *x* is a nonnegative integer.

Operational mode

Use this command to clear statistics for a tunnel interface.

interfaces tunnel <tunx>

Creates a tunnel interface for encapsulating traffic.

Syntax:
`set interfaces tunnel` *tunx*

Syntax:
`delete interfaces tunnel` [*tunx*]

Syntax:
`show interfaces tunnel`

tunx

An identifier for the tunnel interface that you are creating. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
 tunnel tunx {
 }
}
```

Use this command to create a tunnel interface for encapsulating traffic.

Use the `set` form of this command to create a tunnel interface.

Use the `delete` form of this command to delete a tunnel interface.

Use the `show` form of this command to display the tunnel configuration.

interfaces tunnel <tunx> address <address>

Assigns a primary or secondary IP address to a tunnel interface.

Syntax:
`set interfaces tunnel` *tunx* **`address`** { *ipv4_address* | *ipv6_address* }

Syntax:

`delete interfaces tunnel` *tunx* **`address`** [*ipv4_address* | *ipv6_address*]

Syntax:
`show interfaces tunnel` *tunx* **`address`**

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

ipv4_address
An IPv4 address on the interface. The format of the address is *ip_address/prefix* (for example, 192.168.1.77/24).

You can define multiple IPv4 addresses for a single interface by creating multiple **`address`** configuration nodes.

***ipv6*_address**
An IPv6 address on the interface. The format of the address is *ipv6_address/prefix* (for example, 2001:db8:1234::/48).

You can define multiple IPv6 addresses for a single interface by creating multiple **`address`** configuration nodes.

Configuration mode

```
interfaces {
 tunnel tunx {
  address
   ipv4_address
   ipv6_address
 }
}
```

Use this command to assign a primary or secondary IP address to a tunnel interface. At least one address must be configured for the tunnel interface to function.

Note that you cannot use the `set` form of this command to change an existing address; you must delete the address to be changed and create a new one.

Use the `set` form of this command to create an IP address for a tunnel interface.

Use the `delete` form of this command to delete an IP network address for a tunnel interface. At least one address must remain for the tunnel to function.

Use the `show` form of this command to display address configuration for a tunnel interface.

interfaces tunnel <tunx> bfd template

Specifies a bidirectional forwarding detection (BFD) template for a specified interface.

Syntax:
`set interfaces` **`tunnel`** *tunx* **`bfd template`** *template-name*

Syntax:
`delete interfaces` **`tunnel`** *tunx* **`bfd template`** *template-name*

Syntax:
`show interfaces tunnel` **`tunnel`** *tunx* **`bfd template`**

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

template-name
The name of a BFD template to enable add or delete.

Configuration mode

```
interfaces {
 tunnel tunx {
         bfd {
               template my_template
      }
   }
 }
```

Use the `set` form of this command to specify the name of a BFD template.

Use the `delete` form of this command to delete a specified BFD template.

Use the `show` form of this command to display the BFD configuration for the tunnel.

interfaces tunnel <tunx> bridge-group

Adds a specified tunnel interface to a bridge group and sets parameters for the tunnel.

Syntax:
`set interfaces tunnel` *tunx* **`bridge-group`** { **`admin-edge`** | **`auto-edge`** | **`bpdu-guard`** | **`bridge`** *bridge-name* | **`cost`** *cost* | **`network-port`** | **`point-to-point`** *ptp-op-status* | **`priority`** *priority* | **`restrict-tcn`** | **`root-block`** }

Syntax:
`delete interfaces tunnel` *tunx* **`bridge-group`**

Syntax:
`show interfaces tunnel` *tunx* **`bridge-group`**

GRE is the encapsulation type.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

`admin-edge`
Enables the admin-edge mode for Spanning Tree Protocol (STP).

`auto-edge`
Enables the auto-edge mode for STP.

`bpdu-guard`
Enables spanning tree BPDU guard.

`bridge` ***bridge-name***
Name to identify the bridge group.

`cost` ***cost***
Spanning tree port cost (1-65535).

`network-port`
Enables Spanning Tree uni-directional link detection.

`point-to-point` ***ptp-op-status***
Sets the point-to-point operational status. The status is either on or off.

`priority` ***priority***
Sets the port priority for STP. The priority is a number that ranges from 0 through 15.

`restrict-tcn`
Restricts propagation of Spanning Tree topology change notifications.

`root-block`
Restricts the port's ability to take the spanning tree root role.

Configuration mode

```
interfaces {
 tunnel tunx {
  bridge group {
   admin-edge
   auto-edge
   bpdu-guard
   bridge bridge-name
   cost cost
   network-port
   point-to-point ptp-op-status
   priority priority
   restrict-tcn
   root-block
  }
 }
}
```

Use this command to add a tunnel interface to a bridge group.

Use the `set` form of this command to configure the tunnel interface to a bridge group.

Use the `delete` form of this command to remove the tunnel interface.

Use the `show` form of this command to display the current configuration of the tunnel interface.

interfaces tunnel <tunx> description <description>

Describes a tunnel interface.

Syntax:
`set interfaces tunnel` *tunx* **`description`** *description*

Syntax:
`delete interfaces tunnel` *tunx* **`description`**

Syntax:
`show interfaces tunnel` *tunx* **`description`**

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

description
: A brief description of the interface. The default description is an empty character string.

Configuration mode

```
interfaces {
 tunnel tunx {
  description description
 }
}
```

If the description contains spaces, it must be enclosed in double quotation marks.

Use the `set` form of this command to briefly describe a tunnel interface.

Use the `delete` form of this command to delete the description of a tunnel interface.

Use the `show` form of this command to display the description of a tunnel interface.

interfaces tunnel <tunx> disable

Disables a tunnel interface without discarding configuration.

Syntax:

`set interfaces tunnel` *tunx* **`disable`**

Syntax:
`delete interfaces tunnel` *tunx* **`disable`**

Syntax:
`show interfaces tunnel` *tunx*

A tunnel interface is enabled.

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
 tunnel tunx {
  disable
 }
}
```

Use the `set` form of this command to disable a tunnel interface without discarding its configuration.

Use the `delete` form of this command to enable a tunnel interface.

Use the `show` form of this command to display a tunnel interface.

interfaces tunnel <tunx> disable-link-detect

Directs a tunnel interface not to detect a change in the state of the physical link.

Syntax:
`set interfaces tunnel` *tunx* **`disable-link-detect`**

Syntax:
`delete interfaces tunnel` *tunx* **`disable-link-detect`**

Syntax:
`show interfaces tunnel` *tunx*

An interface detects a change in the state of the physical link.

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
 tunnel tunx {
  disable-link-detect
 }
}
```

Use the `set` form of this command to disable detection of a change in the physical state (for example, when a cable is unplugged).

Use the `delete` form of this command to enable detection of a change in the physical state.

Use the `show` form of this command to display the current configuration for the detection of a change in the physical state.

interfaces tunnel <tunx> encapsulation <action>

Sets the encapsulation type for a tunnel interface.

Syntax:
set interfaces tunnel *tunx* **encapsulation** { **gre** | **gre-bridge** | **gre-multipoint** | **ipip** | **ipip6** | **ip6ip6** | **sit** }

Syntax:
set interfaces tunnel *tunx* **encapsulation** { **gre** | **gre-bridge** | **gre-multipoint** | **ipip** | **ipip6** | **ip6ip6** | **sit** }

Syntax:
show interfaces tunnel *tunx* **encapsulation**

GRE is the default encapsulation type.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tunx, where x is a non-negative integer.

gre
Uses GRE to encapsulate transported packets.

gre-bridge
Uses GRE to encapsulate transported packets. This encapsulation type is to be used only for bridging, that is, when the tunnel interface is to be added to a bridge group.

gre-multipoint
Uses multipoint GRE to encapsulate transported packets.

ipip
Uses IP-in-IP to encapsulate transported packets.

ipip6
Uses IP-in-IPv6 to encapsulate transported packets.

ipip
Uses IPv6-in-IPv6 to encapsulate transported packets.

sit
Uses SIT to encapsulate transported packets.

Configuration mode

```
interfaces {
    tunnel tunx {
       encapsulation {
          gre
          gre-bridge
          gre-multipoint
          ipip
          ipip6
          ip6ip6
          sit
        }
     }
```

The Generic Routing Encapsulation (GRE) protocol provides a simple-general purpose mechanism for encapsulating packets from a wide variety of network protocols to be forwarded over another protocol. The original packet (the "passenger" packet) can be one of many arbitrary network protocols-for example a multicast packet, an IPv6 packet, or a non-IP LAN protocol such as AppleTalk, Banyen VINES, or Novell IPX. The delivery protocol can be one of a number of routable IP protocols.

A limitations of a regular GRE encapsulated tunnel is that it cannot be added to a bridge group. GRE for Bridging (using the gre-bridge keyword) provides this ability. It should be used only in cases in which tunnel interfaces are to be included in a bridge group. Refer to AT&T Vyatta Network Operating System Bridging Configuration Guide for further information.

GRE for multipoint (by using the gre-multipoint keyword) is used as part of DMVPN network configurations. Refer to AT&T Vyatta Network Operating System DMVPN Configuration Guide for further information. The main difference between a standard GRE configuration and a GRE-for-multipoint configuration is that, when configuring GRE for multipoint, you do not specify a remote IP. Everything else remains the same.

The IP-in-IP encapsulation protocol is used to tunnel between networks that have different capabilities or policies. For example, an IP-in-IP tunnel can be used to forward multicast packets across a section of a network (such as an IPsec tunnel) that does not support multicast routing. An IP-in-IP tunnel can also be used to influence the routing of the packet, or to deliver a packet to a mobile device using Mobile IP.

The SIT encapsulation is used to tunnel IPv6 across an IPv4 network.

Use the `set` form of this command to set the encapsulation type for a tunnel interface.

Use the `delete` form of this command to delete restore the default encapsulation type, which is GRE, for a tunnel interface.

Use the `show` form of this command to display the encapsulation type for a tunnel interface.

interfaces tunnel <tunx> ipv6

Enables IPv6 for a tunnel.

Syntax:
`set interfaces` **`tunnel`** *tunx* **`ipv6`**

Syntax:
`delete interfaces` **`tunnel`** *tunx* **`ipv6`**

Syntax:
`show interfaces tunnel` *tunx* **`ipv6`**

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
 tunnel tunx {
  ipv6
 }
}
```

Use the `set` form of this command to enable IPv6 for a tunnel.

Use the `delete` form of this command to disable IPv6 for a tunnel.

Use the `show` form of this command to display the configuration for an IPv6 tunnel.

interfaces tunnel <tunx> ipv6 address

Assigns an IPv6 address to a tunnel interface.

Syntax:
`set interfaces tunnel` *tunx* **`ipv6 address`** [**`autoconf`** | **`eui64`** *ipv6prefix* | **`link-local`** *ipv6-address*]

Syntax:
`delete interfaces tunnel` *tunx* **`ipv6 address`** [**`autoconf`** | **`eui64`** *ipv6prefix* | **`link-local`** *ipv6-address*]

Syntax:

`show interfaces` **`tunnel`** *`tunx`* **`ipv6 address`** `[` **`autoconf`** `|` **`eui64`** `|` **`link-local`** `]`

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

`autoconf`
: Generates an IPv6 address by using the SLAAC protocol. Use this option if the interface is performing a "host" function rather than a "router" function. This option can be specified in addition to static IPv6, static IPv4, and IPv4 DHCP addresses on the interface.

`eui64` ***ipv6prefix***
: Specifies a 64-bit IPv6 address prefix that is used to configure an IPv6 address, in EUI-64 format. The system concatenates this prefix with a 64-bit EUI-64 value that is derived from the 48-bit MAC address of the interface.

`link-local` ***ipv6-address***
: Sets the link-local IPv6 address.

Configuration mode

```
interfaces {
 tunnel tunx {
  ipv6 {
   address {
    autoconf
    eui64 ipv6prefix
    link-local ipv6-address
   }
  }
 }
}
```

You can use the **`autoconf`** keyword to direct the system to automatically configure (autoconfigure) the address by using the Stateless Address Autoconfiguration (SLAAC) protocol that is defined in RFC 4862. Alternatively, you can provide an EUI-64 IPv6 address prefix so that the system constructs the IPv6 address.

If you want the system to use SLAAC to acquire addresses on this interface, then in addition to setting this parameter, you must also disable IPv6 forwarding, either globally (by using the `system ipv6 disable-forwarding` command) or specifically on this interface (by using interfaces tunnel <tunx> ipv6 disable-forwarding *(page 37)*).

Use the `set` form of this command to assign an IPv6 address to a tunnel interface.

Use the `delete` form of this command to delete an IPv6 address from a tunnel interface.

Use the `show` form of this command to display the current IPv6 addresses for an interface.

interfaces tunnel <tunx> ipv6 disable

Disables IPv6 on a tunnel interface.

Syntax:

`set interfaces tunnel` *`tunx`* **`ipv6 disable`**

Syntax:

`delete interfaces tunnel` *`tunx`* **`ipv6 disable`**

Syntax:

`show interfaces tunnel` *`tunx`* **`ipv6 disable`**

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
  tunnel tunx {
    ipv6 {
      disable
    }
  }
}
```

By default, IPv6 is enabled on all interfaces. A global command exists which can disable IPv6, namely `set system ipv6 disable`, and this will take precedence over any of the existing per-interface based, IPv6 commands.

IPv6 Forwarding can be disabled via the `set interface tunnel tunx ipv6 disable-forwarding` command, but note that IPv6 traffic can still be terminated on this interface.

IPv6 configuration can be totally disabled via the `set interface tunnel tunx ipv6 disable` command.

Use the `set` form of this command to disable IPv6 on this interface.

Use the `delete` form of this command to enable IPv6 on this interface.

Use the `show` form of this command to display the current IPv6 configuration on this interface.

interfaces tunnel <tunx> ipv6 disable-forwarding

Disables IPv6 forwarding on a tunnel interface.

Syntax:
`set interfaces` **`tunnel`** *`tunx`* **`ipv6 disable-forwarding`**

Syntax:
`delete interfaces` **`tunnel`** *`tunx`* **`ipv6 disable-forwarding`**

Syntax:
`show interfaces tunnel` *`tunx`* **`ipv6 disable-forwarding`**

IPv6 packets are forwarded.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
 tunnel tunx {
  ipv6 {
   disable-forwarding
  }
 }
}
```

You can also disable IPv6 forwarding globally (that is, for all interfaces) by using the `system ipv6 disable-forwarding` command.

Use the `set` form of this command to disable IPv6 packet forwarding on a tunnel interface.

Use the `delete` form of this command to enable IPv6 packet forwarding on a tunnel interface.

Use the `show` form of this command to display the current configuration of IPv6 packet forwarding on a tunnel interface.

interfaces tunnel <tunx> ipv6 dup-addr-detect-transmits <number>

Specifies the number of times to transmit NS packets as part of the DAD process.

Syntax:
`set interfaces` **`tunnel`** *tunx* **`ipv6 dup-addr-detect-transmits`** *number*

Syntax:
`delete interfaces` **`tunnel`** *tunx* **`ipv6 dup-addr-detect-transmits`** [*number*]

Syntax:
`show interfaces` **`tunnel`** *tunx* **`ipv6 dup-addr-detect-transmits`**

One NS packet is transmitted as part of the DAD process.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

number
The number of times to transmit NS packets. The default number of times is 1. Enter 0 to disable DAD.

Configuration mode

```
interfaces {
 tunnel tunx {
  ipv6 {
   dup-addr-detect-transmits number
  }
 }
}
```

Use the `set` form of this command to specify the number of times to transmit Neighbor Solicitation (NS) packets as part of the Duplicate Address Detection (DAD) process.

Use the `delete` form of this command to delete the number of times and use the default number of 1.

Use the `show` form of this command to display the current number of times.

interfaces tunnel <tunx> ipv6 router-advert <action>

Specifies router advertisements (RAs) to be sent from a tunnel interface.

Syntax:
`set interfaces` **`tunnel`** *tunx* **`ipv6 router-advert`** [**`cur-hop-limit`** *limit*] [**`default-lifetime`** *lifetime*] [**`default-preference`** *preference*] [**`link-mtu`** *mtu*] [**`managed-flag`** *state*] [**`max-interval`** *interval*] [**`min-interval`** *interval*] [**`other-config-flag`** *state*] [**`prefix`** *ipv6net* [**`autonomous-flag`** *state* | **`on-link-flag`** *state* | **`preferred-lifetime`** *lifetime* | **`valid-lifetime`** *lifetime*]] [**`reachable-time`** *time*] [**`retrans-timer`** *time*] [**`send-advert`** *state*]

Syntax:
`delete interfaces` **`tunnel`** *tunx* **`ipv6 router-advert`** [**`cur-hop-limit`**] [**`default-lifetime`**] [**`default-preference`**] [**`link-mtu`**] [**`managed-flag`**] [**`max-interval`**] [**`min-interval`**] [**`other-config-flag`**] [**`prefix`** *ipv6net* [**`autonomous-flag`** | **`on-link-flag`** | **`preferred-lifetime`** | **`valid-lifetime`**]] [**`reachable-time`**] [**`retrans-timer`**] [**`send-advert`**]

Syntax:

`show interfaces` **`tunnel`** *tunx* **`ipv6 router-advert`**

Router advertisements are not sent on an interface.

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

`cur-hop-limit` ***limit***
: Limits the Hop Count field of the IP header for outgoing (unicast) IP packets. This limit is placed in the Hop Count field. The limit ranges from 0 through 255. The default limit is 64. A limit of 0 means that the limit is unspecified by the router.

`default-lifetime` ***lifetime***
: Specifies the lifetime, in seconds, associated with the default router. The lifetime either is 0, which indicates that the router is not a default router, or ranges from the interval specified in the **`max-interval`** argument through 9000 (18.2 hours). If not entered, the lifetime is three times the interval specified in the **`max-interval`** argument.

`default-preference` ***preference***
: Specifies the preference associated with the default router. The preference is one of the following:

  `low`: Makes the default router low preference.

  `medium`: Makes the default router medium preference.

  `high`: Makes the default router high preference.

  The default preference is **`medium`**.

`link-mtu` ***mtu***
: Specifies the MTU to be advertised for the link. The MTU either is 0 or ranges from 1280 through the maximum MTU for the type of link, as defined in RFC 2464. The default MTU is 0, which means the MTU is not specified in the router advertisement message. That is because it is expected that the MTU is configured directly on the interface itself and not for routing advertisements. You can configure this option in cases where the link MTU is not well known.

  If the MTU entered here does not match the MTU configured on the interface, the system issues a warning but does not fail.

`managed-flag` ***state***
: Specifies whether to use the administered protocol for address autoconfiguration. The state is either of the following:

  `true`: Specifies that hosts use the administered (stateful) protocol for address autoconfiguration in addition to any addresses autoconfigured using stateless address autoconfiguration.

  `false`: Specifies that hosts use only stateless address autoconfiguration.

  The default state is **`false`**.

`max-interval` ***interval***
: Specifies the maximum time, in seconds, allowed between sending unsolicited multicast router advertisements from the interface. The interval ranges from 4 through 1800. The default interval is 600 (10 minutes).

`min-interval` ***interval***
: Specifies the minimum time, in seconds, allowed between sending unsolicited multicast router advertisements from the interface. The interval ranges from 3 through 0.75 times the interval specified in the **`max-interval`** argument. The default interval is 0.33 times the interval specified in the **`max-interval`** argument.

`other-config-flag` ***state***
: Specifes that the interface uses the administered (stateful) protocol for autoconfiguration of nonaddress information, as defined in RFC 4862. The state is either of the following:

  `true`: Specifies that hosts use the administered protocol for autoconfiguration of nonaddress information.

  `false`: Specifies that hosts use stateless autoconfiguration of nonaddress information.

  The default state is **`false`**.

`prefix` ***ipv6net***

Multi-node. Specifies an IPv6 prefix, in the format *ipv6-address/prefix*, to be advertised on the IPv6 interface.

You can define more than one IPv6 prefix by configuring multiple **prefix** configuration nodes.

autonomous-flag ***state***

Specifies whether the IPv6 prefix can be used for autonomous address configuration as defined in RFC 4862. The state is either of the following:

true: Specifies that the prefix can be used for autonomous address configuration.

false: Specifies that the prefix cannot be used for autonomous address configuration.

The default state is **true**.

on-link-flag ***state***

Specifies whether the IPv6 prefix can be used for on-link determination, as defined in RFC 4862. The state is either of the following:

true: Specifies that the prefix can be used for on-link determination.

false: Specifies that the advertisement makes no statement about on-link or off-link properties of the prefix. For instance, the prefix might be used for address configuration with some addresses belonging to the prefix being on-link and others being off-link.

The default state is **true**.

preferred-lifetime ***lifetime***

Specifies the lifetime, in seconds, that the addresses generated from the IPv6 prefix through Stateless Address Autoconfiguration (SLAAC) is to remain preferred, as defined in RFC 4862. The lifetime is with respect to the time the packet is sent. The lifetime ranges from 1 through 4294967296 plus the **infinity** keyword, which represents forever. (The actual value of **infinity** is a byte in which all bits are set to 1s: 0XFFFFFFFF.) The default lifetime is 604800 (7 days).

valid-lifetime ***lifetime***

Specifies the lifetime, in seconds, that the IPv6 prefix is valid for the purpose of on-link determination, as defined in RFC 4862. The lifetime is with respect to the time the packet is sent. The lifetime ranges from 1 through 4294967296 plus the **infinity** keyword, which represents forever. (The actual value of **infinity** is a byte in which all bits are set to 1s: 0XFFFFFFFF.) The default lifetime is 2592000 (30 days).

reachable-time ***time***

Specifies the length of time, in milliseconds, for which the system assumes a neighbor is reachable after having received a reachability confirmation. This time is used by address resolution and the Neighbor Unreachability Detection algorithm (see Section 7.3 of RFC 2461). The time ranges from 0 through 3600000, where 0 means the reachable time is not specified in the router advertisement message. The default time is 0.

retrans-timer ***time***

Specifies the length of time, in milliseconds, between retransmitted NS messages. This time is used by address resolution and the Neighbor Unreachability Detection algorithm (see Sections 7.2 and 7.3 of RFC 2461). The time ranges from 0 through 4294967295, where 0 means the retransmit time is not specified in the router advertisement message. The default time is 0.

send-advert ***state***

Specifies whether router advertisements are to be sent from this interface. The state is either of the following:

true: Sends router advertisements from this interface.

false: Does not send router advertisements from this interface. If this state is in effect, parameters in this configuration subtree are still used to configure the local implementation parameters.

The default state is **true**.

Configuration mode

```
interfaces {
 tunnel tunx {
  ipv6 {
   router-advert {
    cur-hop-limit limit
    default-lifetime lifetime
    default-preference preference
    link-mtu mtu
    managed-flag state
    max-interval interval
    min-interval interval
    other-config-flag state
    prefix ipv6net {
     autonomous-flag state
     on-link-flag state
     preferred-lifetime lifetime
     valid-lifetime lifetime
    }
    reachable-time time
    retrans-timer time
    send-advert state
   }
  }
 }
}
```

Use this command to configure RAs to be sent from a tunnel interface.

Router advertisements are sent by IPv6 routers to advertise their existence to hosts on the network. IPv6 hosts do not send RAs.

If the **`router-advert`** node of the configuration tree is missing, RAs are not sent. In addition, if IPv6 forwarding is disabled either globally (by using the `system ipv6 disable-forwarding` command) or on the interface (by using the `interfaces tunnel <tunx> ipv6 disable-forwarding` command), RAs are not sent.

Most RA parameters are required by either the Neighbor Discovery (ND) protocol or the Stateless Address Autoconfiguration (SLAAC) protocol. These parameters are used locally for the IPv6 implementation and become part of the RA messages sent to hosts on the network so that they can be configured appropriately.

Use the `set` form of this command to create the **`router-advert`** configuration node and begin to send RAs.

Use the `delete` form of this command to delete the **`router-advert`** configuration node and stop sending RAs.

Use the `show` form of this command to display the current configuration for sending RAs.

interfaces tunnel <tunx> local-interface <interface-name>

Allows tunnel support if the local physical interface designated for a tunnel endpoint is configured using DHCP.

Syntax:
`set interfaces tunnel` *tunx* **`local-interface`** *interface-name*

Syntax:
`delete interfaces tunnel` *tunx* **`local-interface`** *interface-name*

Syntax:
`show interfaces tunnel` *tunx* **`local-interface`**

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

interface-name
The identifier of a physical interface.

Configuration mode

```
interfaces {
    tunnel tunx {
        local-interface interface-name
    }
}
```

For a tunnel to function, both the local and remote endpoints must be configured.

This feature is intended for use on NHRP spoke routes. You can use this command only if NHRP map is configured by using the `interfaces tunnel <tunx> nhrp map` command.

The encapsulation type must be GRE multipoint.

It cannot be used with NHRP hub configurations. The hub end of the NHRP tunnel must be able to reach the DHCP subnet from which the spoke is assigned a local DHCP address.

The tunnel configuration must include either a local physical interface that is configured using DHCP, as specified in this command, or a local IP address, specified by using the `interfaces tunnel tunx local-ip address` command.

The local interface specified in this command must be configured for DHCP.

Use the `set` form of this command to specify the IP address to use as the local endpoint of a tunnel.

Use the `delete` form of this command to delete the local endpoint of a tunnel.

Use the `show` form of this command to display the IP address for the local endpoint of a tunnel.

interfaces tunnel <tunx> local-ip <address>

Specifies the IP address for the local endpoint of a tunnel.

Syntax:
`set interfaces tunnel` *tunx* **`local-ip`** { *ipv4_address* | *ipv6_address* }

Syntax:
`delete interfaces tunnel` *tunx* **`local-ip`** [*ipv4_address* | *ipv6_address*]

Syntax:
`show interfaces tunnel` *tunx* **`local-ip`**

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

ipv4_address
An IPv4 address to use as the tunnel endpoint on the local router. The IP address must already be configured for the interface.

ipv6_address
An IPv6 address to use as the tunnel endpoint on the local router. The IP address must already be configured for the interface.

Configuration mode

```
interfaces {
 tunnel tunx {
  local-ip
   ipv4_address
   ipv6_address
 }
}
```

For a tunnel to function, both the local and remote endpoints must be configured.

Note: You must use different `local-ip` addresses when configuring two or more mGRE tunnels.

The tunnel configuration must include either the local IP address, specified by using this command, or a local physical interface, specified by using the `interfaces tunnel tunx local-interface ifname` command. You can use the local interface option only if NHRP map is configured by using the `interfaces tunnel <tunx> nhrp map` command.

Use the `set` form of this command to specify the IP address to use as the local endpoint of a tunnel.

Use the `delete` form of this command to delete the local endpoint of a tunnel.

Use the `show` form of this command to display the IP address for the local endpoint of a tunnel.

interfaces tunnel <tunx> mtu <mtu>

Sets the MTU size for a tunnel interface.

Syntax:
`set interfaces tunnel` *tunx* **`mtu`** *mtu*

Syntax:
`delete interfaces tunnel` *tunx* **`mtu`** [*mtu*]

Syntax:
`show interfaces tunnel` *tunx* **`mtu`**

Tunnel interface packets have an MTU size of 1476.

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

mtu
: The MTU size, in octets, for the tunnel interface. The size ranges from 68 through 65535. The default size is 1476.

Configuration mode

```
interfaces {
 tunnel tunx {
  mtu mtu
 }
}
```

Use this command to set the size of the maximum transfer unit (MTU) for encapsulated packets that traverse a tunnel.

This MTU size is applied to the packets that are embedded in the encapsulating protocol; it is not the size of the MTU of the “carrier” packets themselves. The MTU size of carrier packets is dictated by the size of the MTU of the physical interface that is transmitting and receiving the tunnel packets.

Use the `set` form of this command to set the MTU size for encapsulated packets traversing the tunnel.

Use the `delete` form of this command to restore the default MTU size, which is 1476, for encapsulated packets.

Use the `show` form of this command to display the MTU size for encapsulated packets.

interfaces tunnel <tunx> multicast

Enables multicast over a specified tunnel.

Syntax:
`set interfaces` **`tunnel`** *tunx* **`multicast`** { **`enable`** | **`disable`** }

Syntax:
`delete interfaces` **`tunnel`** *tunx* **`multicast`**

Syntax:

`show interfaces tunnel` **`tunnel`** *`tunx`* **`multicast`**

tunx

The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

enable

Enable multicast over the tunnel.

disable

Disable multicast over the tunnel.

Configuration mode

```
interfaces {
    tunnel tunx {
        multicast {
            enable
            disable
        }
    }
}
```

Use the `set` form of this command to enable multicast .

Use the `delete` form of this command to disable multicast.

Use the `show` form of this command to display the multicast configuration for this interface.

interfaces tunnel <tunx> parameters ip ignore-df

Configures the tunnel to ignore the DF bit of the payload when encapsulating IP traffic.

Syntax:

`set interfaces tunnel` *`tunx`* **`parameters ip ignore-df`**

Syntax:

`delete interfaces tunnel` *`tunx`* **`parameters ip ignore-df`**

Syntax:

`show interfaces tunnel` *`tunx`* **`parameters ip ignore-df`**

By default, the DF bit of the payload is not ignored.

tunx

The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
 tunnel tunx {
  parameters {
   ip {
    ignore-df
   }
  }
 }
}
```

Note: This command is valid only with GRE and GRE-multipoint encapsulation.

Use the `set` form of this command configure the tunnel to ignore the DF bit of the payload when encapsulating IP traffic.

Use the `delete` form of this command to consider the DF bit of the payload when encapsulating IP traffic.

Use the `show` form of this command to display the value of the ignore-df parameter.

interfaces tunnel <tunx> parameters ip key <key>

Defines an authentication key for a tunnel interface.

Syntax:
`set interfaces tunnel` *tunx* **`parameters ip key`** *key*

Syntax:
`delete interfaces tunnel` *tunx* **`parameters ip key`** [*key*]

Syntax:
`show interfaces tunnel` *tunx* **`parameters ip key`**

No key is configured; authentication is not required.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

key
A key for authenticating the local endpoint to the remote endpoint. The key must match on both ends of the connection for the tunnel to be established.

Configuration mode

```
interfaces {
 tunnel tunx {
  parameters {
   ip {
    key key
   }
  }
 }
}
```

Use this command to provide a simple password-like numeric key for authenticating tunnel endpoints to each other. For the tunnel to be established, keys must be identical at both ends of the tunnel.

Note: This is only valid with GRE and GRE-multipoint encapsulation only.

Use the `set` form of this command to define an authentication key for a tunnel interface.

Use the `delete` form of this command to delete the authentication key for a tunnel interface.

Use the `show` form of this command to display the authentication key for a tunnel interface.

interfaces tunnel <tunx> parameters ip tos <tos>

Specifies the value to write into the Type of Service (ToS) byte of the IP header of a tunnel packet.

Syntax:
`set interfaces tunnel` *tunx* **`parameters ip tos`** *tos*

Syntax:
`delete interfaces tunnel` *tunx* **`parameters ip tos`** [*tos*]

Syntax:
`show interfaces tunnel` *tunx* **`parameters ip tos`**

The default value is 0.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

tos
The value to write into the ToS byte in the IP header of a tunnel packet (the carrier packet). The value ranges from 0 through 99, where 0 means a tunnel packet copies the ToS value from the packet being encapsulated (the passenger packet).

Configuration mode

```
interfaces {
 tunnel tunx {
  parameters {
   ip {
    tos tos
   }
  }
 }
}
```

Use this command to specify the value to write into the 8-bit ToS byte of the IP header for a packet that traverses a tunnel interface. The ToS byte of the IP header of a packet specifies the forwarding behavior to be applied to the packet.

Use the `set` form of this command to specify the ToS value.

Use the `delete` form of this command to restore the default behavior for the ToS byte, that is, the ToS byte of the encapsulated packet is copied into the IP header of the tunnel packet.

Use the `show` form of this command to display the current value for the ToS byte.

interfaces tunnel <tunx> parameters ip ttl <ttl>

Sets the time-to-live (TTL) value to write into the IP header of a tunnel packet.

Syntax:
`set interfaces tunnel` *tunx* **`parameters ip ttl`** *ttl*

Syntax:
`delete interfaces tunnel` *tunx* **`parameters ip ttl`** [*ttl*]

Syntax:
`show interfaces tunnel` *tunx* **`parameters ip ttl`**

The TTL byte of the encapsulated packet is copied into the TTL byte of the IP header of a tunnel packet.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

ttl
The value to write into the TTL field in the IP header of a tunnel packet (the carrier packet). The value ranges from 0 through 255, where 0 means a tunnel packet copies the TTL value from the packet being encapsulated (the passenger packet). The default value is 255.

Configuration mode

```
interfaces {
  tunnel tunx {
   parameters {
    ip {
     ttl ttl
    }
   }
  }
 }
```

The TTL field of the IP header of a packet limits the lifetime of an IP packet and prevents indefinite packet looping.

Use the `set` form of this command to set the TTL value to write into the TTL field of the IP header for a packet that traverses a tunnel interface.

Use the `delete` form of this command to restore the default behavior for the TTL field, that is, the TTL byte of the encapsulated packet is copied into the TTL byte of the IP header of the tunnel packet.

Use the `show` form of this command to display the current TTL value to write into the IP header of a tunnel packet.

interfaces tunnel <tunx> parameters ipv6 encaplimit <limit>

Sets the maximum number of times that a tunnel packet can be encapsulated.

Syntax:
`set interfaces tunnel` *tunx* **`parameters ipv6 encaplimit`** { *0-255* | **`none`** }

Syntax:
`delete interfaces tunnel` *tunx* **`parameters ipv6 encaplimit`** [*0-255* | **`none`**]

Syntax:
`show interfaces tunnel` *tunx* **`parameters ipv6 encaplimit`**

The default limit is 4.

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

0-255
: The limit ranges from 0 through 255.

`none`
: If no limit is specified, the command is disabled.

Configuration mode

```
interfaces {
 tunnel tunx {
  parameters {
   ipv6 {
    encaplimit
     0-255
     none
   }
  }
 }
}
```

Use the `set` form of this command to set the maximum number of times (that is, the number of levels) a packet can be encapsulated.

Use the `delete` form of this command to restore the default maximum number of four times.

Use the `show` form of this command to display the number of times.

interfaces tunnel <tunx> parameters ipv6 flowlabel <flowlabel>

Defines the flow label of the encapsulating IPv6 header.

Syntax:
`set interfaces tunnel` *tunx* **`parameters ipv6 flowlabel`** { *flowlabel* | **`inherit`** }

Syntax:
`delete interfaces tunnel` *tunx* **`parameters ipv6 flowlabel`** [*flowlabel* | **`inherit`**]

Syntax:
`show interfaces tunnel` *tunx* **`parameters ipv6 flowlabel`**

The default flow label is inherit.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

0-0xfffff
The flow label of the encapsulating IPv6 header. The flow label ranges from 0 through 0xfffff.

`inherit`
The default flow label of the encapsulating IPv6 header.

Configuration mode

```
interfaces {
 tunnel tunx {
  parameters {
   ipv6 {
    flowlabel
     0-0xfffff
     inherit
   }
  }
 }
}
```

The flow label field of the encapsulated packet is copied into the flow label field of the IPv6 header of a tunnel packet.

Use the `set` form of this command to specify the flowlabel of the encapsulating IPv6 header.

Use the `delete` form of this command to restore the default flow label, which is the flow label of the encapsulated packet.

Use the `show` form of this command to display the current flow label.

interfaces tunnel <tunx> parameters ipv6 hoplimit <hoplimit>

Sets the hop-limit value to write into the IPv6 header of a tunnel packet.

Syntax:
`set interfaces tunnel` *tunx* **`parameters ipv6 hoplimit`** *0-255*

Syntax:
`delete interfaces tunnel` *tunx* **`parameters ipv6 hoplimit`** [*0-255*]

Syntax:
`show interfaces tunnel` *tunx* **`parameters ipv6 hoplimit`**

The default hop limit is 64.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

0-255
The value to write into the hop-limit field in the IPv6 header of a tunnel packet (the carrier packet). The value ranges from 0 through 255, where 0 means a tunnel packet copies the value from the packet being encapsulated (the passenger packet).

Configuration mode

```
interfaces {
 tunnel tunx {
  parameters {
   ipv6 {
    hoplimit 0-255
   }
  }
 }
}
```

The hop-limit byte of the encapsulated packet is copied into the hop-limit byte of the IPv6 header of a tunnel packet.

Use this command to set the value to write into the hop-limit field of the IPv6 header for a packet that traverses a tunnel interface. The hop-limit field of the IPv6 heder of a packet limits the lifetime of an IPv6 packet and prevents indefinite packet looping.

Use the `set` form of this command to set the hop-limit value to write into the IPv6 header of a tunnel packet.

Use the `delete` form of this command to restore the default behavior for the hoplimit field, that is, the hop-limit byte of the encapsulated packet is copied into the IPv6 header of the tunnel packet.

Use the `show` form of this command to display the current hop-limit value to write into the IPv6 header of a tunnel packet.

interfaces tunnel <tunx> parameters ipv6 tclass <class>

Sets the value to write into the tclass byte of the IPv6 header of a tunnel packet.

Syntax:
`set interfaces tunnel` *tunx* **`parameters ipv6 tclass`** *0-0xff*

Syntax:
`delete interfaces tunnel` *tunx* **`parameters ipv6 tclass`** [*0-0xff*]

Syntax:
`show interfaces tunnel` *tunx* **`parameters ipv6 tclass`**

The default value is inherit.

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

0-0xff
The value to write into the tclass byte in the IPv6 header of a tunnel packet (the carrier packet). The value ranges from 0 through 0xff, where 0 means a tunnel packet copies the tclass byte from the packet being encapsulated (the passenger packet).

inherit
The default traffic class of the encapsulating IPv6 header.

Configuration mode

```
interfaces {
 tunnel tunx {
  parameters {
   ipv6 {
    tclass 0-0xff
   }
  }
 }
}
```

The tclass byte of the encapsulated packet is copied into the tclass byte of the IPv6 header of a tunnel packet. The tclass byte of the IPv6 header of a packet specifies the forwarding behavior to be applied to the packet.

Use the `set` form of this command to set the value to write into the tclass byte of the IPv6 header for a packet that traverses a tunnel interface.

Use the `delete` form of this command to restore the default behavior for the tclass field, that is, the tclass byte of the encapsulated packet is copied into the tclass byte of the IPv6 header of the tunnel packet.

Use the `show` form of this command to display the current tclass value to write into the IPv6 header of a tunnel packet.

interfaces tunnel <tunx> path-mtu-discovery-disable

Disables path Maximum Transmission Unit (MTU) discovery for the local interface used with a specified tunnel.

Syntax:
`set interfaces` **`tunnel`** *`tunx`* **`path-mtu-discovery-disable`**

Syntax:
`delete interfaces` **`tunnel`** *`tunx`* **`path-mtu-discovery-disable`**

Syntax:
`show interfaces tunnel` **`tunnel`** *`tunx`* **`path-mtu-discovery-disable`**

tunx
The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

Configuration mode

```
interfaces {
 tunnel tunx {
   path-mtu-discovery-disable
  }
 }
}
```

Use the `set` form of this command to disable path MTU discovery on the tunnel's local interface.

Use the `delete` form of this command to enable path MTU discovery.

Use the `show` form of this command to display the current path MTU discovery setting for the tunnel interface.

interfaces tunnel <tunx> remote-ip <address>

Sets the IP address for the remote endpoint of a tunnel.

Syntax:
`set interfaces tunnel` *tunx* **`remote-ip`** { *ipv4* | *ipv6* }

Syntax:
`delete interfaces tunnel` *tunx* **`remote-ip`** [*ipv4* | *ipv6*]

Syntax:
`show interfaces tunnel` *tunx* **`remote-ip`**

tunx
: The identifier of a tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

ipv4
: An IPv4 address to use as the tunnel endpoint on the remote router. The IP address must already be configured for the interface.

ipv6
: An IPv6 address to use as the tunnel endpoint on the remote router. The IP address must already be configured for the interface.

Configuration mode

```
interfaces {
 tunnel tunx {
  remote-ip
   ipv4
   ipv6
 }
}
```

Note that the tunnel cannot be established when both the local and remote endpoints are not configured.

Use the `set` form of this command to set the IP address to use as the remote endpoint of a tunnel.

Use the `delete` form of this command to delete the remote endpoint of a tunnel.

Use the `show` form of this command to display the IP address for the remote endpoint of a tunnel.

show interfaces tunnel

Displays the operational status about one tunnel interface or all tunnel interfaces.

Syntax:
`show interfaces tunnel` [*tunx* | **`detail`**]

Information is displayed for all tunnel interfaces.

tunx
: Displays information for the specified tunnel interface. The identifier ranges from tun0 through tun*x*, where *x* is a nonnegative integer.

`detail`
: Displays a detailed status of all tunnel interfaces.

Operational mode

Use this command to display the operational status of one tunnel interface or all tunnel interfaces.

The following example shows how to display the operational status for the GRE tun0 tunnel interface.

```
vyatta@vyatta:~$ show interfaces tunnel
 tun0@NONE: <POINTOPOINT,NOARP,UP,LOWER_UP> mtu 1476 qdisc noqueue
 link/gre 192.168.20.2 peer 192.168.20.3
```

```
inet 192.168.20.1/24 brd 192.168.20.255 scope global tun0
RX:   bytes   packets   errors   dropped   overrun   mcast
  0  0   0   0   0   0
TX:  bytes    packets   errors   dropped   carrier   collisions
  0  0   0   0   0   0
```

VRF support

VRF support for IPsec and GRE

The AT&T Vyatta vRouter provides the following support for IPsec Virtual Tunnel Interface (VTI) and GRE tunnels.

The inner or encapsulated address is configurable for each routing instance, provided that the tunnel interface is bound to a routing instance.

- Assign the IPsec site-to-site configuration to a VTI, which assigns the configuration to a routing-instance.
- Assign the GRE configuration to the interface or interfaces that make up the routing instance.

Command support for VRF routing instances

VRF allows an AT&T Vyatta vRouter to support multiple routing tables, one for each VRF routing instance. Some commands in this guide support VRF and can be applied to particular routing instances.

Use the guidelines in this section to determine correct syntax when adding VRF routing instances to commands. For more information about VRF, refer to AT&T Vyatta Network Operating System Basic Routing Configuration Guide. This guide includes an overview of VRF, VRF configuration examples, information about VRF-specific features, and a list of commands that support VRF routing instances.

Adding a VRF routing instance to a Configuration mode command

For most Configuration mode commands, specify the VRF routing instance at the beginning of a command. Add the appropriate VRF keywords and variable to follow the initial action (**set**, **show**, or **delete**) and before the other keywords and variables in the command.

Example: Configuration mode example: syslog

The following command configures the syslog logging level for the specified syslog host. The command does not include a VRF routing instance, so the command applies to the default routing instance.

```
vyatta@R1# set system syslog host 10.10.10.1 facility all level debug
vyatta@R1# show system syslog
syslog {
    host 10.10.10.1 {
            facility all {
                    level debug
            }
    }
}
```

The following example shows the same command with the VRF routing instance (GREEN) added. Notice that **routing routing-instance GREEN** has been inserted between the basic action (**set** in the example) and the rest of the command. Most Configuration mode commands follow this convention.

```
vyatta@R1# set routing routing-instance GREEN system syslog host 10.10.10.1 facility all
 level debug
vyatta@R1# show routing
routing {
     routing-instance GREEN {
             system {
                     syslog {
                            host 11.12.13.2:514 {
                                     facility all {
                                             level debug
                                     }
                             }
```

```
                        }
                }
        }
}
```

Example: Configuration mode example: SNMP

Some features, such as SNMP, are not available on a per-routing instance basis but can be bound to a specific routing instance. For these features, the command syntax is an exception to the convention of specifying the routing instance at the beginning of Configuration mode commands.

The following example shows how to configure the SNMPv1 or SNMPv2c community and context for the RED and BLUE routing instances. The first two commands specify the RED routing instance as the context for community A and BLUE routing instance as the context for community B. The subsequent commands complete the configuration.

For more information about configuring SNMP, refer to AT&T Vyatta Network Operating System Remote Management Configuration Guide.

```
vyatta@R1# set service snmp community commA context RED
vyatta@R1# set service snmp community commB context BLUE
vyatta@R1# set service snmp view all oid 1
vyatta@R1# set service snmp community commA view all
vyatta@R1# set service snmp community commB view all
vyatta@R1# show service snmp community
 community commA {
        context RED
        view all
 }
 community commB {
        context BLUE
        view all
 }
[edit]
vyatta@vyatta#
```

Adding a VRF routing instance to an Operational mode command

The syntax for adding a VRF routing instance to an Operational mode command varies according to the type of command parameters:

- If the command does not have optional parameters, specify the routing instance at the end of the command.
- If the command has optional parameters, specify the routing instance after the required parameters and before the optional parameters.

Example: Operational mode examples without optional parameters

The following command displays dynamic DNS information for the default routing instance.

```
vyatta@vyatta:~$ show dns dynamic status
```

The following command displays the same information for the specified routing instance (GREEN). The command does not have any optional parameters, so the routing instance is specified at the end of the command.

```
vyatta@vyatta:~$ show dns dynamic status routing-instance GREEN
```

Example: Operational mode example with optional parameters

The following command obtains multicast path information for the specified host (10.33.2.5). A routing instance is not specified, so the command applies to the default routing instance.

```
vyatta@vyatta:~$ mtrace 10.33.2.5 detail
```

The following command obtains multicast path information for the specified host (10.33.2.5) and routing instance (GREEN). Notice that the routing instance is specified before the optional **detail** keyword.

```
vyatta@vyatta:~$ mtrace 10.33.2.5 routing-instance GREEN detail
```

Example: Operational mode example output: SNMP

The following SNMP **show** commands display output for routing instances.

```
vyatta@vyatta:~$ show snmp routing-instance
Routing Instance SNMP Agent is Listening on for Incoming Requests:
Routing-Instance            RDID
-----------------           ----
RED                         5

vyatta@vyatta:~$ show snmp community-mapping
SNMPv1/v2c Community/Context Mapping:
Community                   Context
---------                   -------
commA                       'RED'
commB                       'BLUE'
deva                        'default'


vyatta@vyatta:~$ show snmp trap-target
SNMPv1/v2c Trap-targets:
Trap-target                   Port   Routing-Instance Community
-----------                   ----   ---------------- ---------
1.1.1.1                              'RED'            'test'


vyatta@vyatta:~$ show snmp v3 trap-target
SNMPv3 Trap-targets:
Trap-target                   Port   Protocol Auth Priv Type   EngineID              Routing-
Instance User
-----------                   ----   -------- ---- ---- ----   --------
 ---------------- ----
2.2.2.2                       '162'  'udp'    'md5      'infor                       'BLUE'
         'test'
```

List of Acronyms

Acronym	Description
ACL	access control list
ADSL	Asymmetric Digital Subscriber Line
AH	Authentication Header
AMI	Amazon Machine Image
API	Application Programming Interface
AS	autonomous system
ARP	Address Resolution Protocol
AWS	Amazon Web Services
BGP	Border Gateway Protocol
BIOS	Basic Input Output System
BPDU	Bridge Protocol Data Unit
CA	certificate authority
CCMP	AES in counter mode with CBC-MAC
CHAP	Challenge Handshake Authentication Protocol
CLI	command-line interface
DDNS	dynamic DNS
DHCP	Dynamic Host Configuration Protocol
DHCPv6	Dynamic Host Configuration Protocol version 6
DLCI	data-link connection identifier
DMI	desktop management interface
DMVPN	dynamic multipoint VPN
DMZ	demilitarized zone
DN	distinguished name
DNS	Domain Name System

Acronym	Description
DSCP	Differentiated Services Code Point
DSL	Digital Subscriber Line
eBGP	external BGP
EBS	Amazon Elastic Block Storage
EC2	Amazon Elastic Compute Cloud
EGP	Exterior Gateway Protocol
ECMP	equal-cost multipath
ESP	Encapsulating Security Payload
FIB	Forwarding Information Base
FTP	File Transfer Protocol
GRE	Generic Routing Encapsulation
HDLC	High-Level Data Link Control
I/O	Input/Output
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IEEE	Institute of Electrical and Electronics Engineers
IGMP	Internet Group Management Protocol
IGP	Interior Gateway Protocol
IPS	Intrusion Protection System
IKE	Internet Key Exchange
IP	Internet Protocol
IPOA	IP over ATM
IPsec	IP Security
IPv4	IP Version 4
IPv6	IP Version 6
ISAKMP	Internet Security Association and Key Management Protocol
ISM	Internet Standard Multicast

Acronym	Description
ISP	Internet Service Provider
KVM	Kernel-Based Virtual Machine
L2TP	Layer 2 Tunneling Protocol
LACP	Link Aggregation Control Protocol
LAN	local area network
LDAP	Lightweight Directory Access Protocol
LLDP	Link Layer Discovery Protocol
MAC	medium access control
mGRE	multipoint GRE
MIB	Management Information Base
MLD	Multicast Listener Discovery
MLPPP	multilink PPP
MRRU	maximum received reconstructed unit
MTU	maximum transmission unit
NAT	Network Address Translation
NBMA	Non-Broadcast Multi-Access
ND	Neighbor Discovery
NHRP	Next Hop Resolution Protocol
NIC	network interface card
NTP	Network Time Protocol
OSPF	Open Shortest Path First
OSPFv2	OSPF Version 2
OSPFv3	OSPF Version 3
PAM	Pluggable Authentication Module
PAP	Password Authentication Protocol
PAT	Port Address Translation
PCI	peripheral component interconnect

Acronym	Description
PIM	Protocol Independent Multicast
PIM-DM	PIM Dense Mode
PIM-SM	PIM Sparse Mode
PKI	Public Key Infrastructure
PPP	Point-to-Point Protocol
PPPoA	PPP over ATM
PPPoE	PPP over Ethernet
PPTP	Point-to-Point Tunneling Protocol
PTMU	Path Maximum Transfer Unit
PVC	permanent virtual circuit
QoS	quality of service
RADIUS	Remote Authentication Dial-In User Service
RHEL	Red Hat Enterprise Linux
RIB	Routing Information Base
RIP	Routing Information Protocol
RIPng	RIP next generation
RP	Rendezvous Point
RPF	Reverse Path Forwarding
RSA	Rivest, Shamir, and Adleman
Rx	receive
S3	Amazon Simple Storage Service
SLAAC	Stateless Address Auto-Configuration
SNMP	Simple Network Management Protocol
SMTP	Simple Mail Transfer Protocol
SONET	Synchronous Optical Network
SPT	Shortest Path Tree
SSH	Secure Shell

Acronym	Description
SSID	Service Set Identifier
SSM	Source-Specific Multicast
STP	Spanning Tree Protocol
TACACS+	Terminal Access Controller Access Control System Plus
TBF	Token Bucket Filter
TCP	Transmission Control Protocol
TKIP	Temporal Key Integrity Protocol
ToS	Type of Service
TSS	TCP Maximum Segment Size
Tx	transmit
UDP	User Datagram Protocol
VHD	virtual hard disk
vif	virtual interface
VLAN	virtual LAN
VPC	Amazon virtual private cloud
VPN	virtual private network
VRRP	Virtual Router Redundancy Protocol
WAN	wide area network
WAP	wireless access point
WPA	Wired Protected Access