NAME

ip-l2tp - L2TPv3 static unmanaged tunnel configuration

SYNOPSIS

ip [*OPTIONS*] **l2tp** { *COMMAND* | **help** }

ip l2tp add tunnel
remote *ADDR* **local** *ADDR*
tunnel_id *ID* **peer_tunnel_id** *ID*
[**encap** { **ip** | **udp** }]
[**udp_sport** *PORT*] [**udp_dport** *PORT*]
[**udp_csum** { **on** | **off** }]
[**udp6_csum_tx** { **on** | **off** }]
[**udp6_csum_rx** { **on** | **off** }]
ip l2tp add session [**name** *NAME*]
tunnel_id *ID* **session_id** *ID* **peer_session_id** *ID*
[**cookie** *HEXSTR*] [**peer_cookie** *HEXSTR*]
[**l2spec_type** { **none** | **default** }]
[**seq** { **none** | **send** | **recv** | **both** }]
ip l2tp del tunnel tunnel_id *ID*
ip l2tp del session tunnel_id *ID* **session_id** *ID*
ip l2tp show tunnel [**tunnel_id** *ID*]
ip l2tp show session [**tunnel_id** *ID*.B *] [* **session_id** *ID*]
NAME := *STRING*
ADDR := { *IP_ADDRESS* | **any** }
PORT := { *NUMBER* }
ID := { *NUMBER* }
HEXSTR := { 8 or 16 hex digits (4 / 8 bytes) }

DESCRIPTION

The **ip l2tp** commands are used to establish static, or so-called *unmanaged* L2TPv3 ethernet tunnels. For unmanaged tunnels, there is no L2TP control protocol so no userspace daemon is required - tunnels are manually created by issuing commands at a local system and at a remote peer.

L2TPv3 is suitable for Layer-2 tunneling. Static tunnels are useful to establish network links across IP networks when the tunnels are fixed. L2TPv3 tunnels can carry data of more than one session. Each session is identified by a session_id and its parent tunnel's tunnel_id. A tunnel must be created before a session can be created in the tunnel.

When creating an L2TP tunnel, the IP address of the remote peer is specified, which can be either an IPv4 or IPv6 address. The local IP address to be used to reach the peer must also be specified. This is the address on which the local system will listen for and accept received L2TP data packets from the peer.

L2TPv3 defines two packet encapsulation formats: UDP or IP. UDP encapsulation is most common. IP encapsulation uses a dedicated IP protocol value to carry L2TP data without the overhead of UDP. Use IP encapsulation only when there are no NAT devices or firewalls in the network path.

When an L2TPv3 ethernet session is created, a virtual network interface is created for the session, which must then be configured and brought up, just like any other network interface. When data is passed through the interface, it is carried over the L2TP tunnel to the peer. By configuring the system's routing tables or adding the interface to a bridge, the L2TP interface is like a virtual wire (pseudowire) connected to the peer.

Establishing an unmanaged L2TPv3 ethernet pseudowire involves manually creating L2TP contexts on the local system and at the peer. Parameters used at each site must correspond or no data will be passed. No consistency checks are possible since there is no control protocol used to establish unmanaged L2TP tunnels. Once the virtual network interface of a given L2TP session is configured and enabled, data can be transmitted, even if the peer isn't yet configured. If the peer isn't configured, the L2TP data packets will be discarded by the peer.

To establish an unmanaged L2TP tunnel, use **l2tp add tunnel** and **l2tp add session** commands described in this document. Then configure and enable the tunnel's virtual network interface, as required.

Note that unmanaged tunnels carry only ethernet frames. If you need to carry PPP traffic (L2TPv2) or your peer doesn't support unmanaged L2TPv3 tunnels, you will need an L2TP server which implements the L2TP control protocol. The L2TP control protocol allows dynamic L2TP tunnels and sessions to be established and provides for detecting and acting upon network failures.

ip l2tp add tunnel - add a new tunnel

tunnel_id *ID*
set the tunnel id, which is a 32-bit integer value. Uniquely identifies the tunnel. The value used must match the peer_tunnel_id value being used at the peer.

peer_tunnel_id *ID*
set the peer tunnel id, which is a 32-bit integer value assigned to the tunnel by the peer. The value used must match the tunnel_id value being used at the peer.

remote *ADDR*
set the IP address of the remote peer. May be specified as an IPv4 address or an IPv6 address.

local *ADDR*
set the IP address of the local interface to be used for the tunnel. This address must be the address of a local interface. May be specified as an IPv4 address or an IPv6 address.

encap *ENCAP*
set the encapsulation type of the tunnel.
Valid values for encapsulation are: **udp**, **ip**.

udp_sport *PORT*
set the UDP source port to be used for the tunnel. Must be present when udp encapsulation is selected. Ignored when ip encapsulation is selected.

udp_dport *PORT*
set the UDP destination port to be used for the tunnel. Must be present when udp encapsulation is selected. Ignored when ip encapsulation is selected.

udp_csum *STATE*
(IPv4 only) control if IPv4 UDP checksums should be calculated and checked for the encapsulating UDP packets, when UDP encapsulating is selected. Default is **off**.
Valid values are: **on**, **off**.

udp6_csum_tx *STATE*
(IPv6 only) control if IPv6 UDP checksums should be calculated for encapsulating UDP packets, when UDP encapsulating is selected. Default is **on**.
Valid values are: **on**, **off**.

udp6_csum_rx *STATE*
(IPv6 only) control if IPv6 UDP checksums should be checked for the encapsulating UDP packets, when UDP encapsulating is selected. Default is **on**.
Valid values are: **on**, **off**.

ip l2tp del tunnel - destroy a tunnel

tunnel_id *ID*
set the tunnel id of the tunnel to be deleted. All sessions within the tunnel must be deleted first.

ip l2tp show tunnel - show information about tunnels

tunnel_id *ID*
set the tunnel id of the tunnel to be shown. If not specified, information about all tunnels is printed.

ip l2tp add session - add a new session to a tunnel

name *NAME*
sets the session network interface name. Default is l2tpethN.

tunnel_id *ID*
set the tunnel id, which is a 32-bit integer value. Uniquely identifies the tunnel into which the session will be created. The tunnel must already exist.

session_id *ID*
set the session id, which is a 32-bit integer value. Uniquely identifies the session being created. The value used must match the peer_session_id value being used at the peer.

peer_session_id *ID*
set the peer session id, which is a 32-bit integer value assigned to the session by the peer. The value used must match the session_id value being used at the peer.

cookie *HEXSTR*
sets an optional cookie value to be assigned to the session. This is a 4 or 8 byte value, specified as 8 or 16 hex digits, e.g. 014d3636deadbeef. The value must match the peer_cookie value set at the peer. The cookie value is carried in L2TP data packets and is checked for expected value at the peer. Default is to use no cookie.

peer_cookie *HEXSTR*
sets an optional peer cookie value to be assigned to the session. This is a 4 or 8 byte value, specified as 8 or 16 hex digits, e.g. 014d3636deadbeef. The value must match the cookie value set at the peer. It tells the local system what cookie value to expect to find in received L2TP packets. Default is to use no cookie.

l2spec_type *L2SPECTYPE*
set the layer2specific header type of the session.
Valid values are: **none**, **default**.

seq *SEQ*
controls sequence numbering to prevent or detect out of order packets. **send** puts a sequence number in the default layer2specific header of each outgoing packet. **recv** reorder packets if they are received out of order. Default is **none**.
Valid values are: **none**, **send**, **recv**, **both**.

ip l2tp del session - destroy a session

tunnel_id *ID*
set the tunnel id in which the session to be deleted is located.

session_id *ID*
set the session id of the session to be deleted.

ip l2tp show session - show information about sessions

tunnel_id *ID*
set the tunnel id of the session(s) to be shown. If not specified, information about sessions in all tunnels is printed.

session_id *ID*
set the session id of the session to be shown. If not specified, information about all sessions is printed.

EXAMPLES

Setup L2TP tunnels and sessions

```
site-A:# ip l2tp add tunnel tunnel_id 3000 peer_tunnel_id 4000 \
        encap udp local 1.2.3.4 remote 5.6.7.8 \
        udp_sport 5000 udp_dport 6000
site-A:# ip l2tp add session tunnel_id 3000 session_id 1000 \
        peer_session_id 2000

site-B:# ip l2tp add tunnel tunnel_id 4000 peer_tunnel_id 3000 \
        encap udp local 5.6.7.8 remote 1.2.3.4 \
        udp_sport 6000 udp_dport 5000
```

```
site-B:# ip l2tp add session tunnel_id 4000 session_id 2000 \
        peer_session_id 1000

site-A:# ip link set l2tpeth0 up mtu 1488

site-B:# ip link set l2tpeth0 up mtu 1488
```

Notice that the IP addresses, UDP ports and tunnel / session ids are matched and reversed at each site.

Configure as IP interfaces

The two interfaces can be configured with IP addresses if only IP data is to be carried. This is perhaps the simplest configuration.

```
site-A:# ip addr add 10.42.1.1 peer 10.42.1.2 dev l2tpeth0

site-B:# ip addr add 10.42.1.2 peer 10.42.1.1 dev l2tpeth0

site-A:# ping 10.42.1.2
```

Now the link should be usable. Add static routes as needed to have data sent over the new link.

Configure as bridged interfaces

To carry non-IP data, the L2TP network interface is added to a bridge instead of being assigned its own IP address, using standard Linux utilities. Since raw ethernet frames are then carried inside the tunnel, the MTU of the L2TP interfaces must be set to allow space for those headers.

```
site-A:# ip link set l2tpeth0 up mtu 1446
site-A:# ip link add br0 type bridge
site-A:# ip link set l2tpeth0 master br0
site-A:# ip link set eth0 master br0
site-A:# ip link set br0 up
```

If you are using VLANs, setup a bridge per VLAN and bridge each VLAN over a separate L2TP session. For example, to bridge VLAN ID 5 on eth1 over an L2TP pseudowire:

```
site-A:# ip link set l2tpeth0 up mtu 1446
site-A:# ip link add brvlan5 type bridge
site-A:# ip link set l2tpeth0.5 master brvlan5
site-A:# ip link set eth1.5 master brvlan5
site-A:# ip link set brvlan5 up
```

Adding the L2TP interface to a bridge causes the bridge to forward traffic over the L2TP pseudowire just like it forwards over any other interface. The bridge learns MAC addresses of hosts attached to each interface and intelligently forwards frames from one bridge port to another. IP addresses are not assigned to the l2tpethN interfaces. If the bridge is correctly configured at both sides of the L2TP pseudowire, it should be possible to reach hosts in the peer's bridged network.

When raw ethernet frames are bridged across an L2TP tunnel, large frames may be fragmented and forwarded as individual IP fragments to the recipient, depending on the MTU of the physical interface used by the tunnel. When the ethernet frames carry protocols which are reassembled by the recipient, like IP, this isn't a problem. However, such fragmentation can cause problems for protocols like PPPoE where the recipient expects to receive ethernet frames exactly as transmitted. In such cases, it is important that frames leaving the tunnel are reassembled back into a single frame before being forwarded on. To do so, enable netfilter connection tracking (conntrack) or manually load the Linux netfilter defrag modules at each tunnel endpoint.

```
site-A:# modprobe nf_defrag_ipv4

site-B:# modprobe nf_defrag_ipv4
```

If L2TP is being used over IPv6, use the IPv6 defrag module.

INTEROPERABILITY

Unmanaged (static) L2TPv3 tunnels are supported by some network equipment equipment vendors such as Cisco.

In Linux, L2TP Hello messages are not supported in unmanaged tunnels. Hello messages are used by L2TP clients and servers to detect link failures in order to automate tearing down and reestablishing dynamic tunnels. If a non-Linux peer supports Hello messages in unmanaged tunnels, it must be turned off to interoperate with Linux.

Linux defaults to use the Default Layer2SpecificHeader type as defined in the L2TPv3 protocol specification, RFC3931. This setting must be consistent with that configured at the peer. Some vendor implementations (e.g. Cisco) default to use a Layer2SpecificHeader type of None.

SEE ALSO

ip(8)

AUTHOR

James Chapman <jchapman@katalix.com>