Virtual eXtensible Local Area Networking documentation

The VXLAN protocol is a tunnelling protocol designed to solve the problem of limited VLAN IDs (4096) in IEEE 802.1q. With VXLAN the size of the identifier is expanded to 24 bits (16777216).

VXLAN is described by IETF RFC 7348, and has been implemented by a number of vendors. The protocol runs over UDP using a single destination port. This document describes the Linux kernel tunnel device, there is also a separate implementation of VXLAN for Openvswitch.

Unlike most tunnels, a VXLAN is a 1 to N network, not just point to point. A VXLAN device can learn the IP address of the other endpoint either dynamically in a manner similar to a learning bridge, or make use of statically-configured forwarding entries.

The management of vxlan is done in a manner similar to its two closest neighbors GRE and VLAN. Configuring VXLAN requires the version of iproute2 that matches the kernel release where VXLAN was first merged upstream.

1. Create vxlan device:

```
# ip link add vxlan0 type vxlan id 42 group 239.1.1.1 dev eth1 dstport 4789
```

This creates a new device named vxlan0. The device uses the multicast group 239.1.1.1 over eth1 to handle traffic for which there is no entry in the forwarding table. The destination port number is set to the IANA-assigned value of 4789. The Linux implementation of VXLAN pre-dates the IANA's selection of a standard destination port number and uses the Linux-selected value by default to maintain backwards compatibility.

2. Delete vxlan device:

```
# ip link delete vxlan0
```

3. Show vxlan info:

```
# ip -d link show vxlan0
```

It is possible to create, destroy and display the vxlan forwarding table using the new bridge command.

1. Create forwarding table entry:

```
# bridge fdb add to 00:17:42:8a:b4:05 dst 192.19.0.2 dev vxlan0
```

2. Delete forwarding table entry:

```
# bridge fdb delete 00:17:42:8a:b4:05 dev vxlan0
```

3. Show forwarding table:

```
# bridge fdb show dev vxlan0
```

The following NIC features may indicate support for UDP tunnel-related offloads (most commonly VXLAN features, but support for a particular encapsulation protocol is NIC specific):

- *tx-udp_tnl-segmentation*
- *tx-udp_tnl-csum-segmentation*
  ability to perform TCP segmentation offload of UDP encapsulated frames
- *rx-udp_tunnel-port-offload*
  receive side parsing of UDP encapsulated frames which allows NICs to perform protocol-aware offloads, like checksum validation offload of inner frames (only needed by NICs without protocol-agnostic offloads)

For devices supporting *rx-udp_tunnel-port-offload* the list of currently offloaded ports can be interrogated with *ethtool*:

```
$ ethtool --show-tunnels eth0
Tunnel information for eth0:
  UDP port table 0:
    Size: 4
    Types: vxlan
    No entries
  UDP port table 1:
    Size: 4
    Types: geneve, vxlan-gpe
    Entries (1):
        port 1230, vxlan-gpe
```