NAME

ng_vlan_rotate - IEEE 802.1ad VLAN manipulation netgraph node type

SYNOPSIS

#include <sys/types.h>
#include <netgraph.h>
#include <netgraph/ng_vlan_rotate.h>

DESCRIPTION

The **ng_vlan_rotate** node type manipulates the order of VLAN tags of frames tagged according to the IEEE 802.1ad (an extension of IEEE 802.1Q) standard between different hooks.

Each node has four special hooks, *original*, *ordered*, *excessive*, and *incomplete*.

A frame tagged with an arbitrary number of ETHERTYPE_VLAN, ETHERTYPE_QINQ, and 0x9100 tags received on the *original* hook will be rearranged to a new order of those tags and is sent out the "ordered" hook. After successful processing the *histogram* counter for the observed stack size increments.

If it contains fewer VLANs in the stack than the configured *min* limit, the frame is sent out to the *incomplete* hook and the *incomplete* counter increments.

If there are more VLANs in the stack than the configured *max* limit, the frame is sent out to the *excessive* hook and the *excessive* counter increments.

If the destination hook is not connected, the frame is dropped and the *drops* counter increments.

For Ethernet frames received on the *ordered* hook, the transformation is reversed and is passed to the *original* hook. Please note that this process is identical to the one described above, besides the ordered/original hooks are swapped and the transformation is reversed.

An Ethernet frame received on the *incomplete* or *excessive* hook is forwarded to the *original* hook without any modification.

This node supports only one operation at the moment: Rotation of the VLANs in the stack. Setting the configuration parameter *rot* to a positive value, the stack will roll up by this amount. Negative values will roll down. A typical scenario is setting the value to 1 in order to bring the innermost VLAN tag to the outmost level. Rotation includes the VLAN id, the ether type, and the QOS parameters pcp and cfi. Typical QOS handling refers to the outmost setting, so be careful to keep your QOS intact.

HOOKS

This node type supports the following hooks:

original Typically this hook would be connected to a ng_ether(4) node, using the *lower* hook connected to a carrier network.

ordered Typically this hook would be connected to a ng_vlan(4) type node using the *downstream* hook in order to separate services.

excessive see below.

incomplete Typically those hooks would be attached to a ng_eiface(4) type node using the *ether* hook for anomaly monitoring purposes.

CONTROL MESSAGES

This node type supports the generic control messages, plus the following:

NGM_VLANROTATE_GET_CONF (**getconf**)
Read the current configuration.

NGM_VLANROTATE_SET_CONF (**setconf**)
Set the current configuration.

NGM_VLANROTATE_GET_STAT (**getstat**)
Read the current statistics.

NGM_VLANROTATE_CLR_STAT (**clrstat**)
Zeroize the statistics.

NGM_VLANROTATE_GETCLR_STAT (**getclrstat**)
Read the current statistics and zeroize it in one step.

EXAMPLES

The first example demonstrates how to rotate double or triple tagged frames so that the innermost C-VLAN can be used as service discriminator. The single or double tagged frames (C-VLAN removed) are sent out to an interface pointing to different infrastucture.

```
#!/bin/sh

BNG_IF=ixl3
```

```
VOIP_IF=bge2

ngctl -f- <<EOF
mkpeer ${BNG_IF}: vlan_rotate lower original
name ${BNG_IF}:lower rotate
msg rotate: setconf { min=2 max=3 rot=1 }
mkpeer rotate: vlan ordered downstream
name rotate:ordered services
connect services: ${VOIP_IF} voip lower
msg services: addfilter { vlan=123 hook="voip" }
EOF
```

Now inject the following sample frame on the BNG_IF interface:

```
00:00:00:00:01:01 > 00:01:02:03:04:05,
 ethertype 802.1Q-9100 (0x9100), length 110: vlan 2, p 1,
 ethertype 802.1Q-QinQ, vlan 101, p 0,
 ethertype 802.1Q, vlan 123, p 7,
 ethertype IPv4, (tos 0x0, ttl 64, id 15994, offset 0, flags [none],
  proto ICMP (1), length 84) 192.168.140.101 > 192.168.140.1:
  ICMP echo request, id 40234, seq 0, length 64
```

The frame ejected on the *ordered* hook will look like this:

```
00:00:00:00:01:01 > 00:01:02:03:04:05,
 ethertype 802.1Q (0x8100), length 110: vlan 123, p 7,
 ethertype 802.1Q-9100, vlan 2, p 1,
 ethertype 802.1Q-QinQ, vlan 101, p 0,
 ethertype IPv4, (tos 0x0, ttl 64, id 15994, offset 0, flags [none],
  proto ICMP (1), length 84) 192.168.140.101 > 192.168.140.1:
  ICMP echo request, id 40234, seq 0, length 64
```

Hence, the frame pushed out to the VOIP_IF will have this form:

```
00:00:00:00:01:01 > 00:01:02:03:04:05,
 ethertype 802.1Q-9100, vlan 2, p 1,
 ethertype 802.1Q-QinQ, vlan 101, p 0,
 ethertype IPv4, (tos 0x0, ttl 64, id 15994, offset 0, flags [none],
  proto ICMP (1), length 84) 192.168.140.101 > 192.168.140.1:
  ICMP echo request, id 40234, seq 0, length 64
```

The second example distinguishes between double tagged and single tagged frames.

```
#!/bin/sh

IN_IF=bge1

ngctl -f- <<EOF
mkpeer ${IN_IF}: vlan_rotate lower original
name ${IN_IF}:lower separate
msg separate: setconf { min=1 max=1 rot=0 }
mkpeer separate: eiface incomplete ether
name separate:incomplete untagged
mkpeer separate: eiface ordered ether
name separate:ordered tagged
EOF
```

Setting the *rot* parameter to zero (or omitting it) does not change the order of the tags within the frame. Frames with more VLAN tags are dropped.

SHUTDOWN

This node shuts down upon receipt of a NGM_SHUTDOWN control message, or when all hooks have been disconnected.

SEE ALSO

netgraph(4), ng_eiface(4), ng_ether(4), ng_vlan(4), ngctl(8)

AUTHORS

Lutz Donnerhacke <*lutz@donnerhacke.de*>