author	Ido Schimmel <idosch@nvidia.com>	2022-11-14 10:45:09 +0200
committer	Paolo Abeni <pabeni@redhat.com>	2022-11-15 13:38:11 +0100
commit	9d45921ee4cb364910097e7d1b7558559c2f9fd2 (patch)	
tree	859a5310c89971326887e88b6a6dba24778c1952	
parent	598ab4b12eae70654b6d8af6038e6cdb45f22634 (diff)	
download	linux-9d45921ee4cb364910097e7d1b7558559c2f9fd2.tar.gz	

bridge: switchdev: Fix memory leaks when changing VLAN protocol

The bridge driver can offload VLANs to the underlying hardware either via switchdev or the 8021q driver. When the former is used, the VLAN is marked in the bridge driver with the 'BR_VLFLAG_ADDED_BY_SWITCHDEV' private flag.

To avoid the memory leaks mentioned in the cited commit, the bridge driver will try to delete a VLAN via the 8021q driver if the VLAN is not marked with the previously mentioned flag.

When the VLAN protocol of the bridge changes, switchdev drivers are notified via the 'SWITCHDEV_ATTR_ID_BRIDGE_VLAN_PROTOCOL' attribute, but the 8021q driver is also called to add the existing VLANs with the new protocol and delete them with the old protocol.

In case the VLANs were offloaded via switchdev, the above behavior is both redundant and buggy. Redundant because the VLANs are already programmed in hardware and drivers that support VLAN protocol change (currently only mlx5) change the protocol upon the switchdev attribute notification. Buggy because the 8021q driver is called despite these VLANs being marked with 'BR_VLFLAG_ADDED_BY_SWITCHDEV'. This leads to memory leaks [1] when the VLANs are deleted.

Fix by not calling the 8021q driver for VLANs that were already programmed via switchdev.

```
[1]
unreferenced object 0xffff8881f6771200 (size 256):
  comm "ip", pid 446855, jiffies 4298238841 (age 55.240s)
  hex dump (first 32 bytes):
    00 00 7f 0e 83 88 ff ff 00 00 00 00 00 00 00 00  ................
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
  backtrace:
    [<00000000012819ac>] vlan_vid_add+0x437/0x750
    [<00000000f2281fad>] __br_vlan_set_proto+0x289/0x920
    [<000000000632b56f>] br_changelink+0x3d6/0x13f0
    [<0000000089d25f04>] __rtnl_newlink+0x8ae/0x14c0
    [<00000000f6276baf>] rtnl_newlink+0x5f/0x90
    [<00000000746dc902>] rtnetlink_rcv_msg+0x336/0xa00
    [<000000001c2241c0>] netlink_rcv_skb+0x11d/0x340
    [<0000000010588814>] netlink_unicast+0x438/0x710
    [<00000000e1a4cd5c>] netlink_sendmsg+0x788/0xc40
```

```
    [<00000000e8992d4e>] sock_sendmsg+0xb0/0xe0
    [<00000000621b8f91>] ____sys_sendmsg+0x4ff/0x6d0
    [<000000000ea26996>] ___sys_sendmsg+0x12e/0x1b0
    [<00000000684f7e25>] __sys_sendmsg+0xab/0x130
    [<000000004538b104>] do_syscall_64+0x3d/0x90
    [<0000000091ed9678>] entry_SYSCALL_64_after_hwframe+0x46/0xb0

Fixes: 279737939a81 ("net: bridge: Fix VLANs memory leak")
Reported-by: Vlad Buslov <vladbu@nvidia.com>
Tested-by: Vlad Buslov <vladbu@nvidia.com>
Signed-off-by: Ido Schimmel <idosch@nvidia.com>
Acked-by: Nikolay Aleksandrov <razor@blackwall.org>
Link: https://lore.kernel.org/r/20221114084509.860831-1-idosch@nvidia.com
Signed-off-by: Paolo Abeni <pabeni@redhat.com>
```

Diffstat

-rw-r--r-- net/bridge/br_vlan.c 17

1 files changed, 14 insertions, 3 deletions

```
diff --git a/net/bridge/br_vlan.c b/net/bridge/br_vlan.c
index 6e53dc99140942..9ffd40b8270c17 100644
--- a/net/bridge/br_vlan.c
+++ b/net/bridge/br_vlan.c
@@ -959,6 +959,8 @@ int __br_vlan_set_proto(struct net_bridge *br, __be16 proto,
        list_for_each_entry(p, &br->port_list, list) {
                vg = nbp_vlan_group(p);
                list_for_each_entry(vlan, &vg->vlan_list, vlist) {
+                       if (vlan->priv_flags & BR_VLFLAG_ADDED_BY_SWITCHDEV)
+                               continue;
                        err = vlan_vid_add(p->dev, proto, vlan->vid);
                        if (err)
                                goto err_filt;
@@ -973,8 +975,11 @@ int __br_vlan_set_proto(struct net_bridge *br, __be16 proto,
        /* Delete VLANs for the old proto from the device filter. */
        list_for_each_entry(p, &br->port_list, list) {
                vg = nbp_vlan_group(p);
-               list_for_each_entry(vlan, &vg->vlan_list, vlist)
+               list_for_each_entry(vlan, &vg->vlan_list, vlist) {
+                       if (vlan->priv_flags & BR_VLFLAG_ADDED_BY_SWITCHDEV)
+                               continue;
                        vlan_vid_del(p->dev, oldproto, vlan->vid);
+               }
        }

        return 0;
@@ -983,13 +988,19 @@ err_filt:
        attr.u.vlan_protocol = ntohs(oldproto);
        switchdev_port_attr_set(br->dev, &attr, NULL);

-       list_for_each_entry_continue_reverse(vlan, &vg->vlan_list, vlist)
+       list_for_each_entry_continue_reverse(vlan, &vg->vlan_list, vlist) {
+               if (vlan->priv_flags & BR_VLFLAG_ADDED_BY_SWITCHDEV)
+                       continue;
                vlan_vid_del(p->dev, proto, vlan->vid);
+       }

        list_for_each_entry_continue_reverse(p, &br->port_list, list) {
                vg = nbp_vlan_group(p);
-               list_for_each_entry(vlan, &vg->vlan_list, vlist)
+               list_for_each_entry(vlan, &vg->vlan_list, vlist) {
+                       if (vlan->priv_flags & BR_VLFLAG_ADDED_BY_SWITCHDEV)
```

```
+                               continue;
                        vlan_vid_del(p->dev, proto, vlan->vid);
+               }
        }

        return err;
```