Netdev features mess and how to get out from it alive

Author:
MichaÅ, MirosÅ,aw <mirq-linux@rere.qmqm.pl>

Part I: Feature sets

Long gone are the days when a network card would just take and give packets verbatim. Today's devices add multiple features and bugs (read: offloads) that relieve an OS of various tasks like generating and checking checksums, splitting packets, classifying them. Those capabilities and their state are commonly referred to as netdev features in Linux kernel world.

There are currently three sets of features relevant to the driver, and one used internally by network core:

1. netdev->hw_features set contains features whose state may possibly be changed (enabled or disabled) for a particular device by user's request. This set should be initialized in ndo_init callback and not changed later.
2. netdev->features set contains features which are currently enabled for a device. This should be changed only by network core or in error paths of ndo_set_features callback.
3. netdev->vlan_features set contains features whose state is inherited by child VLAN devices (limits netdev->features set). This is currently used for all VLAN devices whether tags are stripped or inserted in hardware or software.
4. netdev->wanted_features set contains feature set requested by user. This set is filtered by ndo_fix_features callback whenever it or some device-specific conditions change. This set is internal to networking core and should not be referenced in drivers.

Part II: Controlling enabled features

When current feature set (netdev->features) is to be changed, new set is calculated and filtered by calling ndo_fix_features callback and netdev_fix_features(). If the resulting set differs from current set, it is passed to ndo_set_features callback and (if the callback returns success) replaces value stored in netdev->features. NETDEV_FEAT_CHANGE notification is issued after that whenever current set might have changed.

The following events trigger recalculation:

1. device's registration, after ndo_init returned success
2. user requested changes in features state
3. netdev_update_features() is called

ndo_*_features callbacks are called with rtnl_lock held. Missing callbacks are treated as always returning success.

A driver that wants to trigger recalculation must do so by calling netdev_update_features() while holding rtnl_lock. This should not be done from ndo_*_features callbacks. netdev->features should not be modified by driver except by means of ndo_fix_features callback.

Part III: Implementation hints

- ndo_fix_features:

All dependencies between features should be resolved here. The resulting set can be reduced further by networking core imposed limitations (as coded in netdev_fix_features()). For this reason it is safer to disable a feature when its dependencies are not met instead of forcing the dependency on.

This callback should not modify hardware nor driver state (should be stateless). It can be called multiple times between successive ndo_set_features calls.

Callback must not alter features contained in NETIF_F_SOFT_FEATURES or NETIF_F_NEVER_CHANGE sets. The exception is NETIF_F_VLAN_CHALLENGED but care must be taken as the change won't affect already configured VLANs.

- ndo_set_features:

Hardware should be reconfigured to match passed feature set. The set should not be altered unless some error condition happens that can't be reliably detected in ndo_fix_features. In this case, the callback should update netdev->features to match resulting hardware state. Errors returned are not (and cannot be) propagated anywhere except dmesg. (Note: successful return is zero, >0 means silent error.)

Part IV: Features

For current list of features, see include/linux/netdev_features.h. This section describes semantics of some of them.

- Transmit checksumming

For complete description, see comments near the top of include/linux/skbuff.h.

Note: NETIF_F_HW_CSUM is a superset of NETIF_F_IP_CSUM + NETIF_F_IPV6_CSUM. It means that device can fill TCP/UDP-like checksum anywhere in the packets whatever headers there might be.

- Transmit TCP segmentation offload

NETIF_F_TSO_ECN means that hardware can properly split packets with CWR bit set, be it TCPv4 (when NETIF_F_TSO is enabled) or TCPv6 (NETIF_F_TSO6).

- Transmit UDP segmentation offload

NETIF_F_GSO_UDP_L4 accepts a single UDP header with a payload that exceeds gso_size. On segmentation, it segments the payload on gso_size boundaries and replicates the network and UDP headers (fixing up the last one if less than gso_size).

- Transmit DMA from high memory

On platforms where this is relevant, NETIF_F_HIGHDMA signals that ndo_start_xmit can handle skbs with frags in high memory.

- Transmit scatter-gather

Those features say that ndo_start_xmit can handle fragmented skbs: NETIF_F_SG --- paged skbs (skb_shinfo()->frags), NETIF_F_FRAGLIST --- chained skbs (skb->next/prev list).

- Software features

Features contained in NETIF_F_SOFT_FEATURES are features of networking stack. Driver should not change behaviour based on them.

- LLTX driver (deprecated for hardware drivers)

NETIF_F_LLTX is meant to be used by drivers that don't need locking at all, e.g. software tunnels.

This is also used in a few legacy drivers that implement their own locking, don't use it for new (hardware) drivers.

- netns-local device

NETIF_F_NETNS_LOCAL is set for devices that are not allowed to move between network namespaces (e.g. loopback).

Don't use it in drivers.

- VLAN challenged

NETIF_F_VLAN_CHALLENGED should be set for devices which can't cope with VLAN headers. Some drivers set this because the cards can't handle the bigger MTU. [FIXME: Those cases could be fixed in VLAN code by allowing only reduced-MTU VLANs. This may be not useful, though.]

- rx-fcs

This requests that the NIC append the Ethernet Frame Checksum (FCS) to the end of the skb data. This allows sniffers and other tools to read the CRC recorded by the NIC on receipt of the packet.

- rx-all

This requests that the NIC receive all possible frames, including errored frames (such as bad FCS, etc). This can be helpful when sniffing a link with bad packets on it. Some NICs may receive more packets if also put into normal PROMISC mode.

- rx-gro-hw

This requests that the NIC enables Hardware GRO (generic receive offload). Hardware GRO is basically the exact reverse of TSO, and is generally stricter than Hardware LRO. A packet stream merged by Hardware GRO must be re-segmentable by GSO or TSO back to the exact original packet stream. Hardware GRO is dependent on RXCSUM since every packet successfully merged by hardware must also have the checksum verified by hardware.

- hsr-tag-ins-offload

This should be set for devices which insert an HSR (High-availability Seamless Redundancy) or PRP (Parallel Redundancy Protocol) tag automatically.

- hsr-tag-rm-offload

This should be set for devices which remove HSR (High-availability Seamless Redundancy) or PRP (Parallel Redundancy Protocol) tags automatically.

- hsr-fwd-offload

This should be set for devices which forward HSR (High-availability Seamless Redundancy) frames from one port to another in hardware.

- hsr-dup-offload

This should be set for devices which duplicate outgoing HSR (High-availability Seamless Redundancy) or PRP (Parallel Redundancy Protocol) tags automatically frames in hardware.