HOWTO for multiqueue network device support

Section 1: Base driver requirements for implementing multiqueue support

Intro: Kernel support for multiqueue devices

Kernel support for multiqueue devices is always present.

Base drivers are required to use the new alloc_etherdev_mq() or alloc_netdev_mq() functions to allocate the subqueues for the device. The underlying kernel API will take care of the allocation and deallocation of the subqueue memory, as well as netdev configuration of where the queues exist in memory.

The base driver will also need to manage the queues as it does the global netdev->queue_lock today. Therefore base drivers should use the netif_{start|stop|wake}_subqueue() functions to manage each queue while the device is still operational. netdev->queue_lock is still used when the device comes online or when it's completely shut down (unregister_netdev(), etc.).

Section 2: Qdisc support for multiqueue devices

Currently two qdiscs are optimized for multiqueue devices. The first is the default pfifo_fast qdisc. This qdisc supports one qdisc per hardware queue. A new round-robin qdisc, sch_multiq also supports multiple hardware queues. The qdisc is responsible for classifying the skb's and then directing the skb's to bands and queues based on the value in skb->queue_mapping. Use this field in the base driver to determine which queue to send the skb to.

sch_multiq has been added for hardware that wishes to avoid head-of-line blocking. It will cycle though the bands and verify that the hardware queue associated with the band is not stopped prior to dequeuing a packet.

On qdisc load, the number of bands is based on the number of queues on the hardware. Once the association is made, any skb with skb->queue_mapping set, will be queued to the band associated with the hardware queue.

Section 3: Brief howto using MULTIQ for multiqueue devices

The userspace command 'tc,' part of the iproute2 package, is used to configure qdiscs. To add the MULTIQ qdisc to your network device, assuming the device is called eth0, run the following command:

```
# tc qdisc add dev eth0 root handle 1: multiq
```

The qdisc will allocate the number of bands to equal the number of queues that the device reports, and bring the qdisc online. Assuming eth0 has 4 Tx queues, the band mapping would look like:

```
band 0 => queue 0
band 1 => queue 1
band 2 => queue 2
band 3 => queue 3
```

Traffic will begin flowing through each queue based on either the simple_tx_hash function or based on netdev->select_queue() if you have it defined.

The behavior of tc filters remains the same. However a new tc action, skbedit, has been added. Assuming you wanted to route all traffic to a specific host, for example 192.168.0.3, through a specific queue you could use this action and establish a filter such as:

```
tc filter add dev eth0 parent 1: protocol ip prio 1 u32 \
        match ip dst 192.168.0.3 \
        action skbedit queue_mapping 3
```

Author: Alexander Duyck <alexander.h.duyck@intel.com>
Original Author: Peter P. Waskiewicz Jr. <peter.p.waskiewicz.jr@intel.com>