NAME

mac_capab_rings - MAC ring capability

SYNOPSIS

#include <sys/mac_provider.h>

typedef struct mac_capab_rings_s mac_capab_rings_t;

INTERFACE LEVEL

Evolving - This interface is still evolving. API and ABI stability is not guaranteed.

DESCRIPTION

The **MAC_CAPAB_RINGS** capability provides a means to for device drivers to take advantage of the additional resources offered by hardware. There are two primary concepts that this MAC capability relies on: rings and groups.

The *ring* is a logical construct, usually DMA memory, that maps to something that exists in hardware. The ring consists of a number of entries. Each entry in the ring is a descriptor which describes the location in memory of a packet to send or receive and attributes about that packet. These entries are then arranged in a fixed-size circular buffer called a ring that is generally shared between the operating system and the hardware with DMA-backed memory. Most NICs, regardless of their support for this capability, are using something like a ring under the hood. Some vendors may also call the ring a *queue*.

A collection of one or more rings is called a *group*. Each group usually has a collection of filters that can be associated with them. These filters are usually defined in terms of matching something like a MAC address, VLAN, or Ethertype, though more complex filters may exist in hardware. When a packet matches a filter, it will then be directed to those rings.

In the MAC framework, rings and groups are separated into categories based on their purpose: transmitting and receiving. While the MAC framework thinks of transmit and receive rings as different physical constructs, they may map to the same underlying resources in the hardware. The device driver may implement the MAC_CAPAB_RINGS capability for one of transmitting, receiving, or both.

Hardware mapping to Rings, and Groups

There are many different ways that hardware may map to this capability. Consider the following examples:

1. There is hardware that exists that supports a feature commonly called receive side scaling (RSS). With RSS, the hardware has multiple rings and it hashes all incoming traffic and directs that to a different ring. Rings are associated with different interrupts, allowing multiple rings to be

processed in parallel. This would map to a device that has a single group, but multiple rings inside of that group.

2. There is hardware which may have only a single ring, but has support for multiple filters. A common example of this are 1 GbE devices. While the hardware only has one ring, it has support for multiple independent MAC address filters, each of which can be programmed with a single MAC address to receive traffic for. In this case, the driver would map that to a single group with a single ring. However, it would implement the ability to program several filters. While this may not seem useful at first, when virtual NICs are created on top of a physical NIC, the additional hardware filters will be used to avoid putting the device in promiscuous mode.

3. Finally, the third common class of device is one that has many rings, which may be placed into one of many different groups. Each group has its own filtering capabilities. In this world, the device driver would declare support for multiple groups, each of which has its own independent sets of rings.

Filters

The mac_group_info(9S) structure is used to define several different kinds of filters that the group might implement. There are three different classes of filters that exist:

MAC Address
A given frame matches a MAC Address filter if the receive address in the Ethernet Header match the specified MAC address.

VLAN A given frame matches a VLAN filter if it both has an 802.1Q VLAN tag and the specified VLAN matches the VLAN specified in the filter. If the frame’s outer ethertype is not 0x8100, then the filter will not match.

MAC Address and VLAN
A given frame matches a MAC Address and VLAN filter if it matches both the specified MAC address and the specified VLAN. This is constructed as a logical and of the previous two filters. If only one of the two matches, then the frame does not match this filter.

Devices may support many different filter types. If the hardware resources are equal, drivers should prefer to implement the combined MAC Address and VLAN filter, rather than the separate filters.

The MAC framework assumes that the design of the filters when hardware is processing them follows the following rules:

1. When there are multiple filters of the same kind with different addresses, then the hardware will

accept a frame if it matches *ANY* of the specified filters. In other words, if there are two VLAN filters defined, one for VLAN 23 and one for VLAN 42, then if a frame has either VLAN 23 or VLAN 42, then it will be accepted for the group.

2. If multiple different classes of filters are defined, then the hardware should only accept a frame if it passes *ALL* of the filter classes. For example, if there is a MAC address filter and a separate VLAN filter, the hardware will only accept the frame if it passes both sets of filters.

3. If there are multiple different classes of filters and there are multiple filters present in each class, then the driver will accept a packet as long as it matches *ALL* filter classes. However, within a given filter class, it may match *ANY* of the filters.

Device drivers that support the combined MAC and VLAN exact match, should not implement support for the separate MAC and VLAN filters.

The following psuedocode summarizes the behavior for a device that supports independent MAC and VLAN filters. If the hardware only supports a single family of filters, then simply treat that in the psuedocode as though it is always true:

```
for each packet p:
    for each MAC filter m:
        if m matches p’s mac:
            for each VLAN filter v:
                if v matches p’s vlan:
                    accept p for group
```

The following psuedocode summarizes the behavior for a device that supports a combined MAC address and VLAN filter:

```
for each packet p:
    for each filter f:
        if f.mac matches p’s mac and f.vlan matches p’s vlan:
            accept p for group
```

MAC Capability Structure

When the device driver’s mc_getcapab(9E) function entry point is called with the capability requested set to MAC_CAPAB_RINGS, then the value of the capability structure is a pointer to a structure with the following members:

```
uint_t          mr_extensions;
```

```
    uint_t              mr_flags;
    mac_ring_type_t     mr_type;
    mac_groupt_type_t   mr_group_type;
    uint_t              mr_rnum;
    uint_t              mr_gnum;
    mac_get_ring_t      mr_rget;
    mac_get_group_t     mr_gget;
```

If the devices supports the MAC_CAPAB_RINGS capability, then it should first check the *mr_type* member of the structure. This member has the following possible values:

MAC_RING_TYPE_RX
Indicates that this group is for receive rings.

MAC_RING_TYPE_TX
Indicates that this group is for transmit rings.

If neither of these values is specified, then the device driver must return B_FALSE from its mc_getcapab(9E) entry point. Once it has identified the type, it should fill in the capability structure based on the following rules:

mr_extensions The *mr_extensions* member is used to negotiate extensions between the MAC framework and the device driver. The MAC frameowrk will set the value of *mr_extensions* to include all of the currently known extensions. The driver should intersect this list with the set that the driver supports. At this time, no such features are defined and the driver should set this member to **0**.

mr_flags The *mr_flags* member is used to indicate additional capabilities of the ring capability. No such additional capabilities are defined at this time and so this member should be set to **0**.

mr_type The *mr_type* member is used to indicate whether this group is for transmit or receive rings. The *mr_type* member should not be modified by the device driver. It is set by the MAC framework when the driver's mc_getcapab(9E) entry point is called. As indicated above, the driver must check the value to determine which group this is referring to.

mr_group_type
This member is used to indicate the group type. This should be set to MAC_GROUP_TYPE_STATIC, which indicates that there is a static mapping between which rings belong to which group. The number of rings per group may vary on the

group and can be set by the driver.

mr_rnum This indicates the total number of rings that are available from hardware. The number exposed may be less than the number supported in hardware. This is often due to receiving fewer resources such as interrupts.

mr_gnum This indicates the total number of groups that are available from hardware. The number exposed may be less than the number supported in hardware. This is often due to receiving fewer resources such as interrupts.

When working with transmit rings, this value may be zero. In this case, each ring is treated independently and separate groups for each transmit ring are not required.

mr_rget This member is a function that is responsible for filling in information about a group. Please see mr_rget(9E) for information on the function, its signature, and responsibilities.

mr_gget This member is a function that is responsible for filling in information about a group. Please see mr_gget(9E) for information on the function, its signature, and responsibilities.

DRIVER IMPLICATIONS

MAC Callback Entry Points

When a driver implements the MAC_CAPAB_RINGS capability, then it must not implement some of the traditional MAC callbacks. If the driver supports MAC_CAPAB_RINGS for receiving, then it must not implement the mc_unicst(9E) entry point. This is instead handled through the filters that were described earlier. The filter entry points are defined as part of the mac_group_info(9S) structure.

If the driver supports MAC_CAPAB_RINGS for transmitting, then it should not implement the mc_tx(9E) entry point, it will not be used. The MAC framework will instead use the mri_tx(9E) entry point that is part of the mac_ring_info(9S) structure.

Polling on rings

When the MAC_CAPAB_RINGS capability is implemented, then additional functionality for receiving becomes available. A receive ring has the ability to be polled. When the operating system desires to begin polling the ring, it will make a function call into the driver, asking it to receive packets from this ring. When receiving packets while polling, the process is generally identical to that described in the *Receiving Data* section of mac(9E). For more details, see mri_poll(9E).

When the MAC framework wants to enable polling, it will first turn off interrupts through the mi_disable(9E) entry point on the driver. The driver must ensure that there is proper serialization between the interrupt enablement, interrupt disablement, the interrupt handler for that ring, and the mri_poll(9E) entry point. For more information on the locking, see the discussions in mri_poll(9E) and mi_disable(9E).

Updated callback functions

When using rings, two of the primary functions that were used change. First, the mac_rx(9F) function should be replaced with the mac_ring_rx(9F) function. Secondly, the mac_tx_update(9F) function should be replaced with the mac_tx_ring_update(9F) function.

Interrupt and Ring Mapping

Drivers often vary the number of rings that they expose based on the number of interrupts that exist. When a driver only supports a single group, there is often no reason to have more rings than interrupts. However, most hardware supports a means of having multiple rings tie to the same interrupt. Drivers then tie the rings in different groups to the same interrupts and therefore when an interrupt is triggered, iterate over all of the rings.

Filter Management

As part of general operation, the device driver will be asked to add various filters to groups. The MAC framework does not keep track of the assigned filters in such a way that after a device reset that they'll be given to the driver again. Therefore, it is recommended that the driver keep track of all filters it has assigned such that they can be reinstated after a device reset of some kind.

For more information, see the *TX STALL DETECTION, DEVICE RESETS, AND FAULT MANAGEMENT* section of mac(9E).

Broadcast, Multicast, and Promiscuous Mode

Rings and groups are currently designed to emphasize and enhance the receipt of filtered, unicast frames. This means that special handling is required when working with broadcast traffic, multicast traffic, and enabling promiscuous mode. This only applies to receive groups and rings.

By default, only the first group with index zero, sometimes called the default group, should ever be programmed to receive broadcast traffic. This group should always be programmed to receive broadcast traffic, the same way that the broader device is programmed to always receive broadcast traffic when the MAC_CAPAB_RINGS capability has not been negotiated.

When multicast addresses are assigned to the device through the mc_multicst(9E) entry point, those should also be assigned to the first group.

Similarly, when enabling promiscuous mode, the driver should only enable promiscuous traffic to be received by the first group.

No other groups or rings should every receive broadcast, multicast, or promiscuous mode traffic.

SEE ALSO

mac(9E), mc_getcapab(9E), mc_multicst(9E), mc_tx(9E), mc_unicst(9E), mi_disable(9E), mr_gaddring(9E), mr_gget(9E), mr_gremring(9E), mr_rget(9E), mri_poll(9E), mac_ring_rx(9F), mac_rx(9F), mac_tx_ring_update(9F), mac_tx_update(9F), mac_group_info(9S)