Linux kernel driver for Compute Engine Virtual Ethernet (gve):

Supported Hardware

The GVE driver binds to a single PCI device id used by the virtual Ethernet device found in some Compute Engine VMs.

Field	Value	Comments
Vendor ID	*0x1AE0*	Google
Device ID	*0x0042*	
Sub-vendor ID	*0x1AE0*	Google
Sub-device ID	*0x0058*	
Revision ID	*0x0*	
Device Class	*0x200*	Ethernet

PCI Bars

The gVNIC PCI device exposes three 32-bit memory BARS: - Bar0 - Device configuration and status registers. - Bar1 - MSI-X vector table - Bar2 - IRQ, RX and TX doorbells

Device Interactions

The driver interacts with the device in the following ways:

- Registers
    - A block of MMIO registers
    - See gve_register.h for more detail
- Admin Queue
    - See description below
- Reset
    - At any time the device can be reset
- Interrupts
    - See supported interrupts below
- Transmit and Receive Queues
    - See description below

Descriptor Formats

GVE supports two descriptor formats: GQI and DQO. These two formats have entirely different descriptors, which will be described below.

Registers

All registers are MMIO.

The registers are used for initializing and configuring the device as well as querying device status in response to management interrupts.

Endianness

- Admin Queue messages and registers are all Big Endian.
- GQI descriptors and datapath registers are Big Endian.
- DQO descriptors and datapath registers are Little Endian.

Admin Queue (AQ)

The Admin Queue is a PAGE_SIZE memory block, treated as an array of AQ commands, used by the driver to issue commands to the device and set up resources.The driver and the device maintain a count of how many commands have been submitted and executed. To issue AQ commands, the driver must do the following (with proper locking):

1. Copy new commands into next available slots in the AQ array
2. Increment its counter by he number of new commands
3. Write the counter into the GVE_ADMIN_QUEUE_DOORBELL register
4. Poll the ADMIN_QUEUE_EVENT_COUNTER register until it equals the value written to the doorbell, or until a timeout.

The device will update the status field in each AQ command reported as executed through the ADMIN_QUEUE_EVENT_COUNTER register.

Device Resets

A device reset is triggered by writing 0x0 to the AQ PFN register. This causes the device to release all resources allocated by the driver, including the AQ itself.

Interrupts

The following interrupts are supported by the driver:

Management Interrupt

The management interrupt is used by the device to tell the driver to look at the GVE_DEVICE_STATUS register.

The handler for the management irq simply queues the service task in the workqueue to check the register and acks the irq.

Notification Block Interrupts

The notification block interrupts are used to tell the driver to poll the queues associated with that interrupt.

The handler for these irqs schedule the napi for that block to run and poll the queues.

GQI Traffic Queues

GQI queues are composed of a descriptor ring and a buffer and are assigned to a notification block.

The descriptor rings are power-of-two-sized ring buffers consisting of fixed-size descriptors. They advance their head pointer using a __be32 doorbell located in Bar2. The tail pointers are advanced by consuming descriptors in-order and updating a __be32 counter. Both the doorbell and the counter overflow to zero.

Each queue's buffers must be registered in advance with the device as a queue page list, and packet data can only be put in those pages.

Transmit

gve maps the buffers for transmit rings into a FIFO and copies the packets into the FIFO before sending them to the NIC.

Receive

The buffers for receive rings are put into a data ring that is the same length as the descriptor ring and the head and tail pointers advance over the rings together.

DQO Traffic Queues

- Every TX and RX queue is assigned a notification block.
- TX and RX buffers queues, which send descriptors to the device, use MMIO doorbells to notify the device of new descriptors.
- RX and TX completion queues, which receive descriptors from the device, use a "generation bit" to know when a descriptor was populated by the device. The driver initializes all bits with the "current generation". The device will populate received descriptors with the "next generation" which is inverted from the current generation. When the ring wraps, the current/next generation are swapped.
- It's the driver's responsibility to ensure that the RX and TX completion queues are not overrun. This can be accomplished by limiting the number of descriptors posted to HW.
- TX packets have a 16 bit completion_tag and RX buffers have a 16 bit buffer_id. These will be returned on the TX completion and RX queues respectively to let the driver know which packet/buffer was completed.

Transmit

A packet's buffers are DMA mapped for the device to access before transmission. After the packet was successfully transmitted, the buffers are unmapped.

Receive

The driver posts fixed sized buffers to HW on the RX buffer queue. The packet received on the associated RX queue may span multiple descriptors.