DPDK

DATA PLANE DEVELOPMENT KIT

Xen Guide

Release 16.11.0

November 13, 2016

CHAPTER

ONE

DPDK XEN BASED PACKET-SWITCHING SOLUTION

1.1 Introduction

DPDK provides a para-virtualization packet switching solution, based on the Xen hypervisor's Grant Table, Note 1, which provides simple and fast packet switching capability between guest domains and host domain based on MAC address or VLAN tag.

This solution is comprised of two components; a Poll Mode Driver (PMD) as the front end in the guest domain and a switching back end in the host domain. XenStore is used to exchange configure information between the PMD front end and switching back end, including grant reference IDs for shared Virtio RX/TX rings, MAC address, device state, and so on. XenStore is an information storage space shared between domains, see further information on XenStore below.

The front end PMD can be found in the DPDK directory lib/ librte_pmd_xenvirt and back end example in examples/vhost_xen.

The PMD front end and switching back end use shared Virtio RX/TX rings as para- virtualized interface. The Virtio ring is created by the front end, and Grant table references for the ring are passed to host. The switching back end maps those grant table references and creates shared rings in a mapped address space.

The following diagram describes the functionality of the DPDK Xen Packet- Switching Solution.

Note 1 The Xen hypervisor uses a mechanism called a Grant Table to share memory between domains (http://wiki.xen.org/wiki/Grant Table).

A diagram of the design is shown below, where "gva" is the Guest Virtual Address, which is the data pointer of the mbuf, and "hva" is the Host Virtual Address:

In this design, a Virtio ring is used as a para-virtualized interface for better performance over a Xen private ring when packet switching to and from a VM. The additional performance is gained by avoiding a system call and memory map in each memory copy with a XEN private ring.

1.2 Device Creation

1.2.1 Poll Mode Driver Front End

- Mbuf pool allocation:

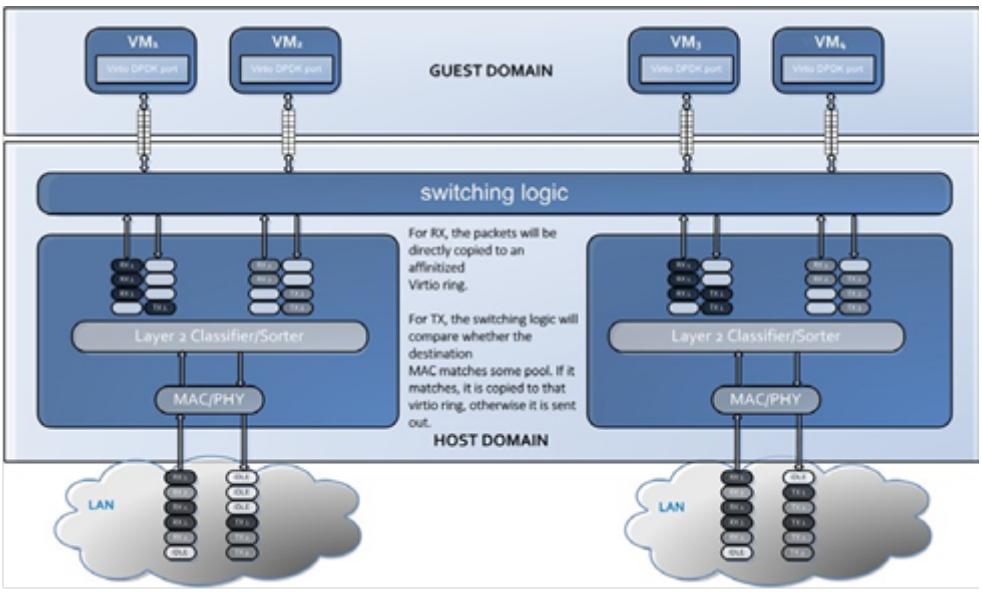

Fig. 1.1: Functionality of the DPDK Xen Packet Switching Solution.

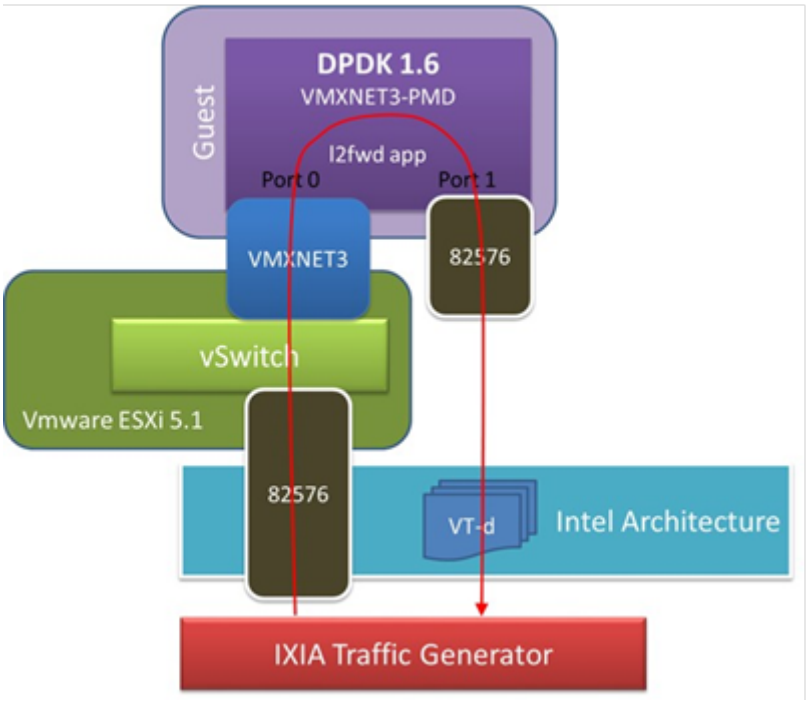

Fig. 1.2: DPDK Xen Layout

To use a Xen switching solution, the DPDK application should use rte_mempool_gntalloc_create() to reserve mbuf pools during initialization. rte_mempool_gntalloc_create() creates a mempool with objects from memory allocated and managed via gntalloc/gntdev.

The DPDK now supports construction of mempools from allocated virtual memory through the rte_mempool_xmem_create() API.

This front end constructs mempools based on memory allocated through the xen_gntalloc driver. rte_mempool_gntalloc_create() allocates Grant pages, maps them to continuous virtual address space, and calls rte_mempool_xmem_create() to build mempools. The Grant IDs for all Grant pages are passed to the host through XenStore.

- Virtio Ring Creation:

  The Virtio queue size is defined as 256 by default in the VQ_DESC_NUM macro. Using the queue setup function, Grant pages are allocated based on ring size and are mapped to continuous virtual address space to form the Virtio ring. Normally, one ring is comprised of several pages. Their Grant IDs are passed to the host through XenStore.

  There is no requirement that this memory be physically continuous.

- Interrupt and Kick:

  There are no interrupts in DPDK Xen Switching as both front and back ends work in polling mode. There is no requirement for notification.

- Feature Negotiation:

  Currently, feature negotiation through XenStore is not supported.

- Packet Reception & Transmission:

  With mempools and Virtio rings created, the front end can operate Virtio devices, as it does in Virtio PMD for KVM Virtio devices with the exception that the host does not require notifications or deal with interrupts.

XenStore is a database that stores guest and host information in the form of (key, value) pairs. The following is an example of the information generated during the startup of the front end PMD in a guest VM (domain ID 1):

```
xenstore -ls /local/domain/1/control/dpdk
0_mempool_gref="3042,3043,3044,3045"
0_mempool_va="0x7fcbc6881000"
0_tx_vring_gref="3049"
0_rx_vring_gref="3053"
0_ether_addr="4e:0b:d0:4e:aa:f1"
0_vring_flag="3054"
...
```

Multiple mempools and multiple Virtios may exist in the guest domain, the first number is the index, starting from zero.

The idx#_mempool_va stores the guest virtual address for mempool idx#.

The idx#_ether_adder stores the MAC address of the guest Virtio device.

For idx#_rx_ring_gref, idx#_tx_ring_gref, and idx#_mempool_gref, the value is a list of Grant references. Take idx#_mempool_gref node for example, the host maps those Grant references to a continuous virtual address space. The real Grant reference information is stored in this virtual address space, where (gref, pfn) pairs follow each other with -1 as the terminator.

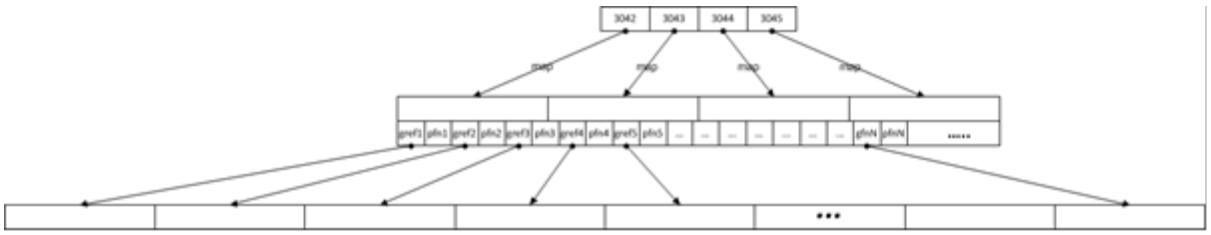

Fig. 1.3: Mapping Grant references to a continuous virtual address space

After all gref# IDs are retrieved, the host maps them to a continuous virtual address space. With the guest mempool virtual address, the host establishes 1:1 address mapping. With multiple guest mempools, the host establishes multiple address translation regions.

1.2.2 Switching Back End

The switching back end monitors changes in XenStore. When the back end detects that a new Virtio device has been created in a guest domain, it will:

1. Retrieve Grant and configuration information from XenStore.
2. Map and create a Virtio ring.
3. Map mempools in the host and establish address translation between the guest address and host address.
4. Select a free VMDQ pool, set its affinity with the Virtio device, and set the MAC/ VLAN filter.

1.2.3 Packet Reception

When packets arrive from an external network, the MAC?VLAN filter classifies packets into queues in one VMDQ pool. As each pool is bonded to a Virtio device in some guest domain, the switching back end will:

1. Fetch an available entry from the Virtio RX ring.
2. Get gva, and translate it to hva.
3. Copy the contents of the packet to the memory buffer pointed to by gva.

The DPDK application in the guest domain, based on the PMD front end, is polling the shared Virtio RX ring for available packets and receives them on arrival.

1.2.4 Packet Transmission

When a Virtio device in one guest domain is to transmit a packet, it puts the virtual address of the packet's data area into the shared Virtio TX ring.

The packet switching back end is continuously polling the Virtio TX ring. When new packets are available for transmission from a guest, it will:

1. Fetch an available entry from the Virtio TX ring.
2. Get gva, and translate it to hva.

3. Copy the packet from hva to the host mbuf's data area.
4. Compare the destination MAC address with all the MAC addresses of the Virtio devices it manages. If a match exists, it directly copies the packet to the matched VIrtio RX ring. Otherwise, it sends the packet out through hardware.

Note: The packet switching back end is for demonstration purposes only. The user could implement their switching logic based on this example. In this example, only one physical port on the host is supported. Multiple segments are not supported. The biggest mbuf supported is 4KB. When the back end is restarted, all front ends must also be restarted.

1.3 Running the Application

The following describes the steps required to run the application.

1.3.1 Validated Environment

Host:

Xen-hypervisor: 4.2.2

Distribution: Fedora release 18

Kernel: 3.10.0

Xen development package (including Xen, Xen-libs, xen-devel): 4.2.3

Guest:

Distribution: Fedora 16 and 18

Kernel: 3.6.11

1.3.2 Xen Host Prerequisites

Note that the following commands might not be the same on different Linux* distributions.

- Install xen-devel package:

```
yum install xen-devel.x86_64
```

- Start xend if not already started:

```
/etc/init.d/xend start
```

- Mount xenfs if not already mounted:

```
mount -t xenfs none /proc/xen
```

- Enlarge the limit for xen_gntdev driver:

```
modprobe -r xen_gntdev
modprobe xen_gntdev limit=1000000
```

Note: The default limit for earlier versions of the xen_gntdev driver is 1024. That is insufficient to support the mapping of multiple Virtio devices into multiple VMs, so it is necessary to enlarge

the limit by reloading this module. The default limit of recent versions of xen_gntdev is 1048576. The rough calculation of this limit is:

limit=nb_mbuf# * VM#.

In DPDK examples, nb_mbuf# is normally 8192.

1.3.3 Building and Running the Switching Backend

1. Edit config/common_linuxapp, and change the default configuration value for the following two items:

```
CONFIG_RTE_LIBRTE_XEN_DOM0=y
CONFIG RTE_LIBRTE_PMD_XENVIRT=n
```

2. Build the target:

```
make install T=x86_64-native-linuxapp-gcc
```

3. Ensure that RTE_SDK and RTE_TARGET are correctly set. Build the switching example:

```
make -C examples/vhost_xen/
```

4. Load the Xen DPDK memory management module and preallocate memory:

```
insmod ./x86_64-native-linuxapp-gcc/build/lib/librte_eal/linuxapp/xen_dom0/rte_dom0_mm.ko
echo 2048> /sys/kernel/mm/dom0-mm/memsize-mB/memsize
```

Note: On Xen Dom0, there is no hugepage support. Under Xen Dom0, the DPDK uses a special memory management kernel module to allocate chunks of physically continuous memory. Refer to the *DPDK Getting Started Guide* for more information on memory management in the DPDK. In the above command, 4 GB memory is reserved (2048 of 2 MB pages) for DPDK.

5. Load uio_pci_generic and bind one Intel NIC controller to it:

```
modprobe uio_pci_generic
python tools/dpdk-devbind.py -b uio_pci_generic 0000:09:00:00.0
```

In this case, 0000:09:00.0 is the PCI address for the NIC controller.

6. Run the switching back end example:

```
examples/vhost_xen/build/vhost-switch -c f -n 3 --xen-dom0 -- -p1
```

Note: The -xen-dom0 option instructs the DPDK to use the Xen kernel module to allocate memory.

Other Parameters:

- -vm2vm

  The vm2vm parameter enables/disables packet switching in software. Disabling vm2vm implies that on a VM packet transmission will always go to the Ethernet port and will not be switched to another VM

- -Stats

The Stats parameter controls the printing of Virtio-net device statistics. The parameter specifies the interval (in seconds) at which to print statistics, an interval of 0 seconds will disable printing statistics.

1.3.4 Xen PMD Frontend Prerequisites

1. Install xen-devel package for accessing XenStore:

   ```
   yum install xen-devel.x86_64
   ```

2. Mount xenfs, if it is not already mounted:

   ```
   mount -t xenfs none /proc/xen
   ```

3. Enlarge the default limit for xen_gntalloc driver:

   ```
   modprobe -r xen_gntalloc
   modprobe xen_gntalloc limit=6000
   ```

Note: Before the Linux kernel version 3.8-rc5, Jan 15th 2013, a critical defect occurs when a guest is heavily allocating Grant pages. The Grant driver allocates fewer pages than expected which causes kernel memory corruption. This happens, for example, when a guest uses the v1 format of a Grant table entry and allocates more than 8192 Grant pages (this number might be different on different hypervisor versions). To work around this issue, set the limit for gntalloc driver to 6000. (The kernel normally allocates hundreds of Grant pages with one Xen front end per virtualized device). If the kernel allocates a lot of Grant pages, for example, if the user uses multiple net front devices, it is best to upgrade the Grant alloc driver. This defect has been fixed in kernel version 3.8-rc5 and later.

1.3.5 Building and Running the Front End

1. Edit config/common_linuxapp, and change the default configuration value:

   ```
   CONFIG_RTE_LIBRTE_XEN_DOM0=n
   CONFIG_RTE_LIBRTE_PMD_XENVIRT=y
   ```

2. Build the package:

   ```
   make install T=x86_64-native-linuxapp-gcc
   ```

3. Enable hugepages. Refer to the *DPDK Getting Started Guide* for instructions on how to use hugepages in the DPDK.

4. Run TestPMD. Refer to *DPDK TestPMD Application User Guide* for detailed parameter usage.

   ```
   ./x86_64-native-linuxapp-gcc/app/testpmd -c f -n 4 --vdev="net_xenvirt0,mac=00:00:00:00:00
   testpmd>set fwd mac
   testpmd>start
   ```

   As an example to run two TestPMD instances over 2 Xen Virtio devices:

   ```
   --vdev="net_xenvirt0,mac=00:00:00:00:00:11" --vdev="net_xenvirt1;mac=00:00:00:00:00:22"
   ```

1.3.6 Usage Examples: Injecting a Packet Stream Using a Packet Generator

Loopback Mode

Run TestPMD in a guest VM:

```
./x86_64-native-linuxapp-gcc/app/testpmd -c f -n 4 --vdev="net_xenvirt0,mac=00:00:00:00:00:11"
testpmd> set fwd mac
testpmd> start
```

Example output of the vhost_switch would be:

```
DATA:(0) MAC_ADDRESS 00:00:00:00:00:11 and VLAN_TAG 1000 registered.
```

The above message indicates that device 0 has been registered with MAC address 00:00:00:00:00:11 and VLAN tag 1000. Any packets received on the NIC with these values is placed on the device's receive queue.

Configure a packet stream in the packet generator, set the destination MAC address to 00:00:00:00:00:11, and VLAN to 1000, the guest Virtio receives these packets and sends them out with destination MAC address 00:00:00:00:00:22.

Inter-VM Mode

Run TestPMD in guest VM1:

```
./x86_64-native-linuxapp-gcc/app/testpmd -c f -n 4 --vdev="net_xenvirt0,mac=00:00:00:00:00:11"
```

Run TestPMD in guest VM2:

```
./x86_64-native-linuxapp-gcc/app/testpmd -c f -n 4 --vdev="net_xenvirt0,mac=00:00:00:00:00:22"
```

Configure a packet stream in the packet generator, and set the destination MAC address to 00:00:00:00:00:11 and VLAN to 1000. The packets received in Virtio in guest VM1 will be forwarded to Virtio in guest VM2 and then sent out through hardware with destination MAC address 00:00:00:00:00:33.

The packet flow is:

packet generator->Virtio in guest VM1->switching backend->Virtio in guest VM2->switching backend->wire