dpdk virtio\_recv\_pkts\_inorder함수로 packet이 들어온다

virtio\_dev\_rx\_split함수로 packet이 virtqueue에 분배된다.

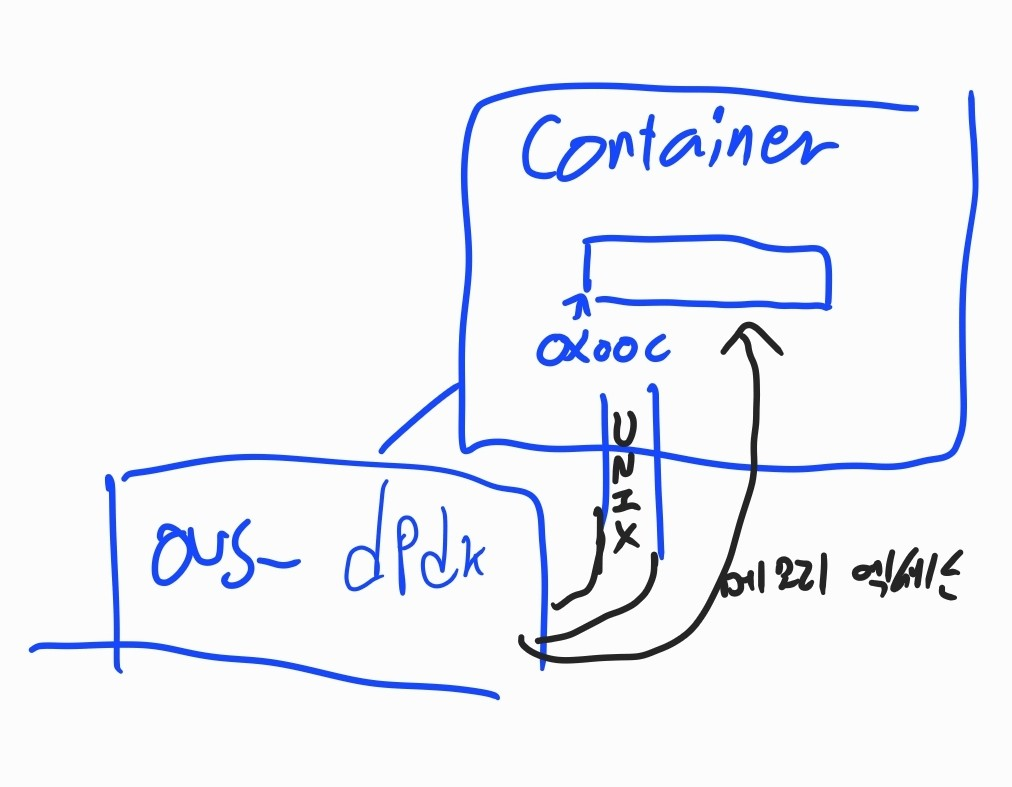
<https://www.programmersought.com/article/9291230023/>

모든 control information은 UNIX socket을 통해 exchanged.

DMA를 위한 memory map information, kick events (queue다찼다 가져가) 그리고 interrupt info that need to be initiated when data is populated into the virtio queue.

data channel은 DMA로 implement되어있다. 클라이언트의 virtio-net driver은 allocates a portion of the memory for the queue of virtio. 즉 virtionet driver가 virtioqueue 할당해놓는다. (virtio ring cache로)

virtio standara를 따르는 queue이다. 컨테이너에선 이 address를 UNIX socket을 통해 OVSDPDK에 전달해준다. DPDK는 virtio queue structure를(vrings ring structure) 같은 주소에 적용해서 client의 memory에 read하고 write할 수 있다.



OVS-DPDK가 container에게 패킷을 전달하면 이 패킷들은 UNIX-socket의 tx traffic으로 잡힌다.

근데 상대방에게 virtio queue에 패킷이 다 복사되었다고 알릴 방법이 필요하다.

그래서 socket에 쓸 수 있는 system call이 필요하고, 상대방은 interrupt operation을 handle해야한다. 근데 상대방에게 unwilling to receive an interrupt 하다고 special flag를 세울 수 있다.

근데 안받는건 temporary or fixed query virtio queue모드에서만 가능하다.

컨테이너의 퍼포먼스를 고려해서 DPDK로 data packet을 process할 수 있다. OVS-DPDK는 packet을 client에게 높은 속도로 보낼 수 있다. 동시에 virtio queue의 send와 receive cache 수는 256 ~1024 로 정해져있다. 그래서 client는 패킷을 엄청빨리 프로세싱해야한다. 그래서 DPDK의 PMD driver를 써서 client port에서 지속적으로 poll 해서 packet processing을 해야 한다.

qemu -netdev type=tap,script=/etc/kvm/kvm-ifup,id=net0,vhost=on **\**

-device virtio-net-pci,netdev=net0

netdev를 tap으로 했을 때 fast zero copy guest to host kernel data path가 존재한다.

<http://www.virtualopensystems.com/en/solutions/guides/snabbswitch-qemu/>

series of ioctl로 한다.

Map to Vring events using eventfd

fd 보내는쪽이 SCM\_RIGHTS flag

[root@overcloud-0 ~]# lsof -nn | grep vhuc26fd3c6-4b | awk '{print $1}' | uniq

로 이 소켓이 OVS를 위해만들어진 것임을 확인 가능

한쪽이 copy해서 data message를 virtio ring of shared memory에 넣으면 반대는 두 option:

1.linux kernel의 NAPI나 DPDK의 PMD polling queue처럼 새 메시지가 notification 없이 obtained.

2. non-queue polling일땐 notify 받아야한다.

이 notify 받을때가 vhu unix socket으로 eventfd file descriptor data를 받아서 control channel이 user space에서 interrupted 된다. write operation of socket이 syscall을 필요로 하고, PMD 가 kernel space에서 spend some time을 하게 만든다. client는 VRING\_AVAIL\_F\_NO\_INTERRUPT flag를 해서 interrupt notification을 안받을 수 있다.

2.4 Virtqueues

The big data transfer mechanism of the virtio device is named virtqueue virtual queue. Each device can have multiple virtqueues or no

Virtqueue queue. The 16-bit queue size parameter specifies the number of members in the queue and also the total size of the queue.

Each virtual queue has three parts:

Descriptor Table - Descriptor Table

Available Ring - Available Ring

Used Ring - Used Ring

1. struct virtq\_avail *{*
2. *#define VIRTQ\_AVAIL\_F\_NO\_INTERRUPT 1*
3. *le16 flags;*
4. *le16 idx;*
5. *le16 ring[ /\* Queue Size \*/ ];*
6. *le16 used\_event; /\* Only if VIRTIO\_F\_EVENT\_IDX \*/*
7. *}*;
8. The driver uses the available ring to provide a send buffer to the device. Each of these rings points to the beginning of a chain of descriptors. The available ring can only be written by the driver and read by the device.
9. The idx member instructs the driver where to place the next descriptor entry in the ring member (not exceeding the queue length). It starts from 0.
10. The traditional standard [Virtio PCI Draft] defines this structure as vring\_avail and the macro definition is named
11. VRING\_AVAIL\_F\_NO\_INTERRUPT, but the essential structure is still the same

dpdk-18.08/drivers/net/virtio/virtio\_ring.h

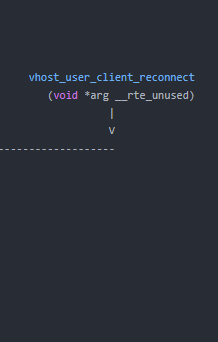
 dpdk-18.08/lib/librte\_vhost/vhost.h <- vhost\_virtqueue

dpdk-18.08/lib/librte\_vhost/vhost\_user.h <- UNIX socket에서 주고받는 메시지.

dpdk-18.08/lib/librte\_vhost/vhost\_user.c <- 메시지 핸들,read\_vhost\_message,send\_vhost\_message

When creating the vhost user interface, the Open vSwitch controls the DPDK to register a new vhost-user UNIX socket. The path to the socket is vhost\_sock\_dir plus netdev->name plus the device's dev->vhost\_id.

By setting the RTE\_VHOST\_USER\_CLIENT flag, OVS can request the creation of a client mode for the vhost user socket.



<https://www.programmersought.com/article/76418222079/>

Netdev\_dpdk\_vhost\_construct is defined in the file openvswitch-2.9.2/lib/netdev-dpdk.c

 rte\_vhost\_driver\_register.   dpdk-18.08/lib/librte\_vhost/socket.c

The netdev\_dpdk\_vhost\_construct function calls rte\_vhost\_driver\_start. Defined in dpdk-18.08/lib/librte\_vhost/socket.c

Vhost\_user\_add\_connection then executes the vhost\_user\_read\_cb function, which in turn calls the vhost\_user\_msg\_handler function to process the received message.

**Virtio tells DPDK shared memory of virtio queues memory address**

The DPDK uses the function vhost\_user\_set\_vring\_addr to convert the virtio descriptor, used ring and available ring address into the DPDK's own address space.

dpdk-18.08/lib/librte\_vhost/vhost\_user.c

즉, vhu로 virt queue address가 전달되면 그걸 translate\_ring address를 통해서 dpdk가 쓰는 메모리 어드레스로 변환한다.

### OVS DPDK sends packets to the client and sends packets

The function that sends packets to the client in the OVS DPDK is \_\_netdev\_dpdk\_vhost\_send, located in the file openvswitch-2.9.2/lib/netdev-dpdk.c.

이제 파일 보내는영역이나오네

The OVS send program will still try to send VHOST\_ENQ\_RETRY\_NUM (default 8) times after the space is used up. If in the first attempt to send, no packets are sent successfully (no packets are written to the shared memory ring), or if the VHOST\_ENQ\_RETRY\_NUM macro is exceeded, the remaining packets will be discarded (batch can be up to 32 Composition of data packets).

아 이게 8번까지 보내보고 , 처음보내려할때 아무것도 성공적으로 못가면 (memory ring에 못쓰거나) VHOST\_ENQ\_RETRY\_NUM 만큼 보냈는데 초과했다면, packet은 discarded되고, (최대 32개 batch로 discard)

openvswitch-2.9.2/lib/netdev-dpdk.c. 이부분의 2097번째 줄에 dp\_packet\_delete(pkts[i]) 가 존재ㅔ한다.

### Client receive interrupt handling

When the OVS DPDK fills a new packet into the virtio ring, there are two scenarios:

* The client is not polling its queue and needs to inform its arrival of the new packet;
* The client is polling the queue and does not need to tell the arrival of new packets.

polling 과 interrupt 섞어쓰는 형식 -> NAPI 방식은 어떻게된다 소개

client가 polling하고 있는경우

When the client polls, the CPU's caches cache utilization is very high, avoiding extra latency. The appropriate processes in the host and client are running, further reducing latency. In addition, when the host sends an interrupt IRQ to the client, it needs to write to the UNIX socket (system call), which is very time consuming and adds extra delay and overhead.

As part of the NFV application, the advantage of running a DPDK in a client is the way its PMD drives process traffic. The PMD driver works in polling mode, shutting down the system interrupt, and the OVS DPDK no longer needs to send interrupt notifications to the client. OVS DPDK saves the operation of writing UNIX sockets and does not require kernel system calls. The OVS DPDK is always running in user space, and the client can also eliminate the interruptions caused by the control channel and run quickly

UNIX socket에 쓸일을 없애준다.

If the VRING\_AVAIL\_F\_NO\_INTERRUPT flag is not set, the client can receive an interrupt. Interrupts to the client are implemented through callfd and the operating system's eventfd component.

The client's OS can enable or disable interrupts. When the client disables the interrupt of the virtio interface, the virtio-net driver is implemented by the macro VRING\_AVAIL\_F\_NO\_INTERRUPT. This macro is defined in both DPDK and QEMU:

1. [root@overcloud-0 SOURCES]# grep VRING\_AVAIL\_F\_NO\_INTERRUPT -R | grep def
2. dpdk-18.08/drivers/net/virtio/virtio\_ring.h:#define VRING\_AVAIL\_F\_NO\_INTERRUPT 1
3. dpdk-18.08/drivers/crypto/virtio/virtio\_ring.h:#define VRING\_AVAIL\_F\_NO\_INTERRUPT 1
4. [root@overcloud-0 qemu]# grep AVAIL\_F\_NO\_INTERRUPT -R -i | grep def
5. qemu-3.0.0/include/standard-headers/linux/virtio\_ring.h:#define VRING\_AVAIL\_F\_NO\_INTERRUPT 1
6. qemu-3.0.0/roms/seabios/src/hw/virtio-ring.h:#define VRING\_AVAIL\_F\_NO\_INTERRUPT 1
7. qemu-3.0.0/roms/ipxe/src/include/ipxe/virtio-ring.h:#define VRING\_AVAIL\_F\_NO\_INTERRUPT 1
8. qemu-3.0.0/roms/seabios-hppa/src/hw/virtio-ring.h:#define VRING\_AVAIL\_F\_NO\_INTERRUPT 1
9. qemu-3.0.0/roms/SLOF/lib/libvirtio/virtio.h:#define VRING\_AVAIL\_F\_NO\_INTERRUPT 1

OS에서 Interrupt를 disable할 수 있다. 이 flag가 not set ==0 이면 client는 receive interrupt할 수 있다.

Once the VRING\_AVAIL\_F\_NO\_INTERRUPT flag bit in vq->avail->flags is set, the DPDK is instructed not to send an interrupt to the client. Dpdk-18.08/lib/librte\_vhost/vhost.h

하지만 1로 된 순간, DPDK는 interrupt안보낸다.

여기가 보내는 코드다.

1. \_\_netdev\_dpdk\_vhost\_send(struct netdev \*netdev, int qid,

이런 코드가존재한다.

The rte\_vhost\_enqueue\_burst function comes from the DPDK vhost library.

grep rte\_vhost\_enqueue\_burst dpdk-16.08/ -R

아 이커맨드로 찾는거구나

이런;;

어쨌던 netdev\_dpdk\_vhost\_send에서 tx packet을 보내고 try가 너무 많으면 packet을 delete한다.

dpdk-18.08/lib/librte\_vhost/rte\_vhost.h

rte\_vhost\_enqueue\_burst는 여기존재한다.

1. 493 \* This function adds buffers to the virtio devices RX virtqueue. Buffers can
2. 494 \* be received from the physical port or from another virtual device. A packet
3. 495 \* count is returned to indicate the number of packets that were successfully
4. 496 \* added to the RX queue.
5. 497 \* @param vid
6. 498 \* vhost device ID
7. 499 \* @param queue\_id
8. 500 \* virtio queue index in mq case
9. 501 \* @param pkts
10. 502 \* array to contain packets to be enqueued
11. 503 \* @param count
12. 504 \* packets num to be enqueued
13. 505 \* @return
14. 506 \* num of packets enqueued
15. 507 \*/
16. 508 uint16\_t rte\_vhost\_enqueue\_burst(int vid, uint16\_t queue\_id,
17. 509 struct rte\_mbuf \*\*pkts, uint16\_t count)

이제 virtio devices RX virtqueue에 내가 가진 버퍼를 넣어주는 것인데 ( memcpy가 아니지. shared mem에 그냥 buffer의 포인터를 넣어주는거니까)

buffer들은 received from the physical port or from another virtual device야.

parameter가 차례대로있는데, 이중 3번째 pkts 라는것이 버퍼 배열이야. 이게 physical port에서 온 패킷인가봐.

1. virtio\_dev\_rx(struct virtio\_net \*dev, uint16\_t queue\_id,
2. 888 struct rte\_mbuf \*\*pkts, uint32\_t count)

virtio\_dev\_rx\_packed function and the virtio\_dev\_rx\_split function send the packet to the client and decide whether to send an interrupt notification (write system call) depending on the settings.  
dpdk-18.08/lib/librte\_vhost/virtio\_net.c

packed랑 split이랑 둘다 packet을 client에게 보내는 함수고, unix socket에 쓸지 말지는 (interrupt보낼지는) 세팅에 달려있다.

In the virtio\_dev\_rx function:

1. 913 count = RTE\_MIN((uint32\_t)MAX\_PKT\_BURST, count);
2. 914 if (count == 0)
3. 915 goto out;

The number of transmitted packets is set to the smaller of the MAX\_PKT\_BURST macro and the number of free entries (count).

이제 virtio\_dev\_rx에서 packet을 보내는 수량은 MAX\_PKT\_BURST와 free entries의 개수중 적은 수에 달려있다.

 value of the used index is increased according to the number of packets sent.

1. virtio\_dev\_rx\_packed(struct virtio\_net \*dev, struct vhost\_virtqueue \*vq,
2. 835 struct rte\_mbuf \*\*pkts, uint32\_t count)
3. virtio\_dev\_rx\_split(struct virtio\_net \*dev, struct vhost\_virtqueue \*vq,
4. 783 struct rte\_mbuf \*\*pkts, uint32\_t count)

packed 나 split에서 value of used index가 증가한다. packet보낸 수량에 따라.

The data packet is copied to the client's memory via the function copy\_mbuf\_to\_desc. Finally, depending on the configuration, decide whether to send interrupt notifications, see the functions vhost\_vring\_call\_split and vhost\_vring\_call\_packed.

클라이언트의 메모리로 copy\_mbuf\_to\_desc로 복사가 이뤄진다. (virt queue에 넣는거 다음에 copy를 해줘야 복사가 되는건가보다?)

rte\_vhost\_enqueue\_burst

(vhost device id, virtio queue index, pkts (physical port나 virtual device에서 받은 array to contain packets to be enqueued) , count ( number of packets to be enqueued) 가

virtio\_dev\_rx도 같은 걸파라미터로 받고

이건

virtio\_dev\_rx\_packed에 virtqueue vq 가 두번째 파라미터, pkts가 세번째 파라미터, 4번째가 count

그리고 count가 세팅되고 (number of free entries 랑 max\_pkt\_burst랑)

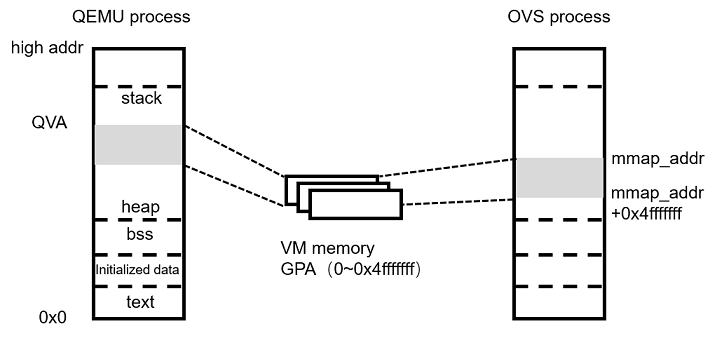
그러면 virtio\_dev\_rx\_packed나 split에도 똑같이 들어오는데,

copy\_mbuf\_to\_desc로는 vhost device id, virt queue, pkts[pkt\_idx] (들어온 패킷중 특정버퍼) , buf\_vec,nr\_vec, num\_buffers) 파라미터가 넘겨진다.

이건

<https://github.com/ceph/dpdk/blob/master/lib/librte_vhost/virtio_net.c>

에 있는데, virtqueue는 vring\_desc로 받고, pkts[pkt\_idx]는 rte\_mbuf로 받아서

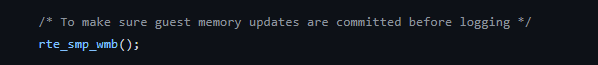
pkts[pkt\_idx]->next에 있는걸 계속 계속 mbuf\_avail에 받아오고, rte\_memcpy로 desc\_addr (gpa\_to\_vva 로 받아온 (guest physical address to vSwitch virtual address?) 주소에 memcopy를 한다. 이제 virtqueue의 address로 얻어온건 GPA로 표현되어있는데, OS는 virtual address에 써야지 인식한다. QVA는 QEMU virtual address

그다음에 vhost\_log\_write로 어디부터 어디를 얼만큼 카피했는지 넘겨준다.

|  |
| --- |
| cpy\_len = RTE\_MIN(desc\_avail, mbuf\_avail); |
| rte\_memcpy((void )((uintptr\_t)(desc\_addr + desc\_offset)), |
| rte\_pktmbuf\_mtod\_offset(m, void \*, mbuf\_offset), |
| cpy\_len); |
| vhost\_log\_write(dev, desc->addr + desc\_offset, cpy\_len); |
| mbuf\_avail -= cpy\_len; |
| mbuf\_offset += cpy\_len; |
| desc\_avail -= cpy\_len; |
| desc\_offset += cpy\_len; |

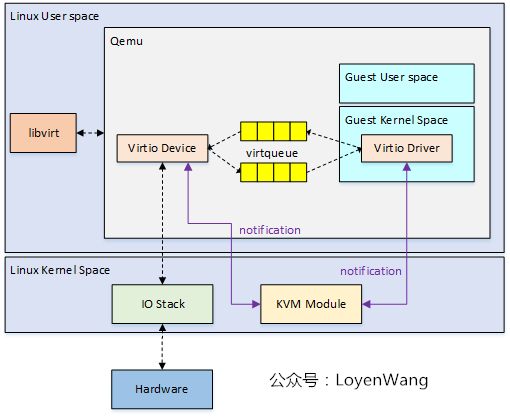
desc는 virtqueue,

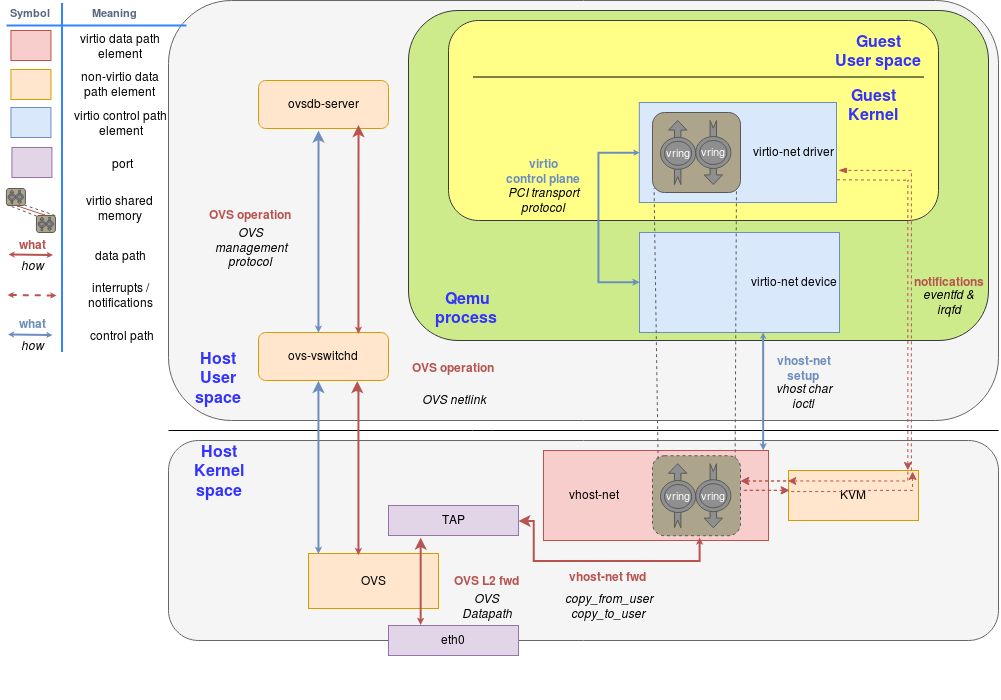
mbuf는 physical port에게서 패킷받아온 큐부분



log write는 음;;몰라

근데 어디부터 어디까지를 썼다를 남겨놔서 복구할수있는거같음.





<https://www.programmersought.com/article/21096302265/>

# 2. Data path processing

The data path is implemented in lib/librte\_vhost/virtio\_net.c of DPDK. Although the code looks very verbose, most of it deals with various features and hardware offloading functions, but the main logic is very simple.

The main function responsible for the sending and receiving of data packets is:

uint16\_t rte\_vhost\_enqueue\_burst(int vid, uint16\_t queue\_id, struct rte\_mbuf \*\*pkts, uint16\_t count)

//Data packets flow from OVS to VM

uint16\_t rte\_vhost\_dequeue\_burst(int vid, uint16\_t queue\_id,

struct rte\_mempool \*mbuf\_pool, struct rte\_mbuf \*\*pkts, uint16\_t count)

//Data packets flow from VM to OVS

The specific sending process can be summarized as follows: if OVS sends a data packet to the VM, the corresponding vhost port reads the available buffer address in the avail ring, converts to VVA, and then copies the data packet, and sends eventfd to notify the VM after the copy is completed; If the VM sends to OVS, on the contrary, copy from the data packet buffer in the VM to the mbuf data structure of DPDK. Let's post a code comment below, don't care about the iommu and iova inside, those are new features of vhost-user, which can be understood as iova is GPA

Batch processing is widely used in the network functions implemented by software, and the number of batches can be set by yourself, but it is generally customary to batch process up to 32 data packets.

