Open vSwitch

December 2020

Enabling asynchronous Para-virtual IO in OVS

Sunil Pai G
Intel

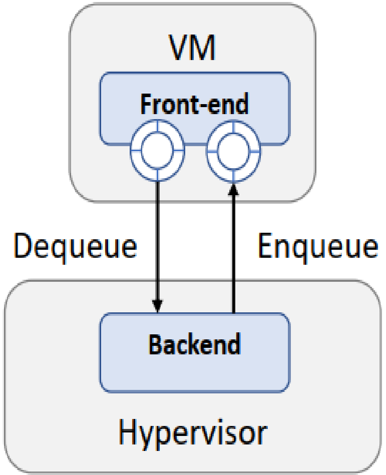
VirtIO

- Para-virtual I/O is a virtualization technique to enhance VM I/O performance.

- VirtIO is a standard of para-virtual I/O, which consists of VirtIO front-end in VM and backend in hypervisor.

- Back-end communicates with front-end by copying packet buffers between hypervisor's and VM's memory

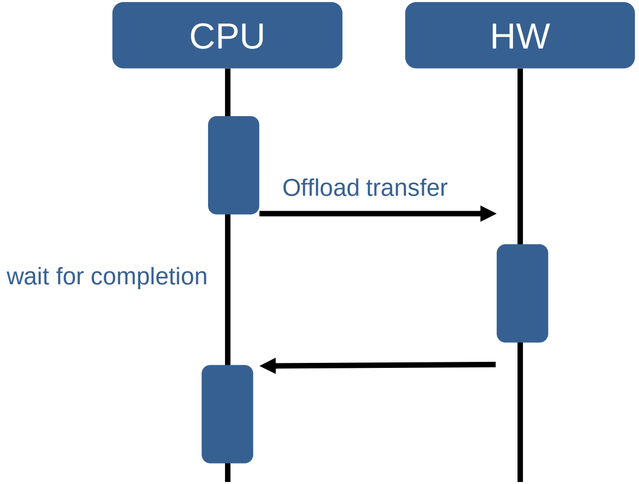
- **Copying large bulk of data** between backend and frontend becomes a **hotspot**



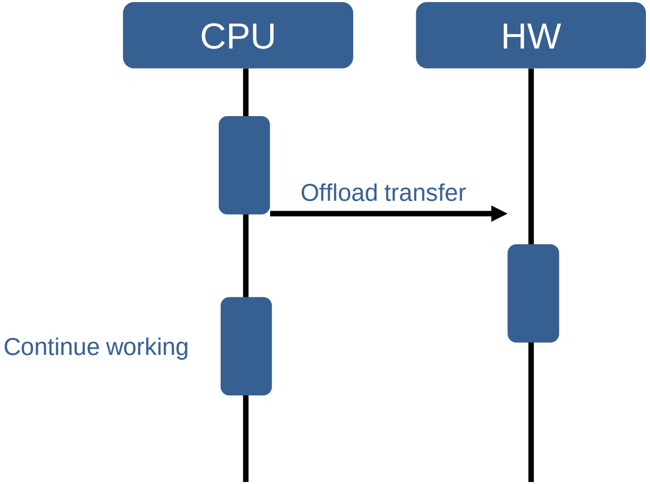
src: https://www.dpdk.org/wp-content/uploads/sites/35/2018/12/JiayuHu_Accelerating_paravirtio_with_CBDMA.pdf

Offloading modes:

Synchronous mode:



Asynchronous mode:



DPDK API's

vHost async API's (vHost Library)

- rte_vhost_submit_enqueue_burst /* enqueue packets */
- rte_vhost_poll_enqueue_completed /* query send status */

/* operation callbacks */

- struct rte_vhost_async_channel_ops {

```
        transfer_data(...);
        check_completed_copies(...);
};
```

/* tie the callback and threshold to vid, qid pair */

- rte_vhost_async_channel_register
- rte_vhost_async_channel_unregister

- set RTE_VHOST_USER_ASYNC_COPY flag :
  rte_vhost_driver_register

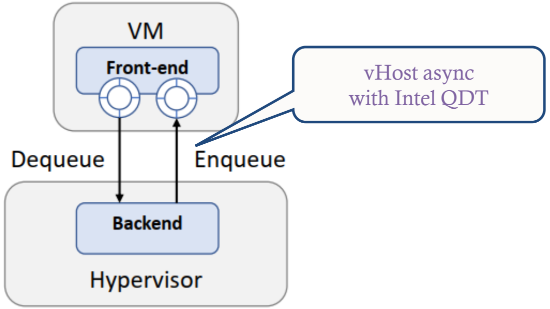
HW: Intel® QuickData Technology (QDT) (IOAT PMD)

- rte_rawdev_info_get
- rte_rawdev_configure
- rte_rawdev_start

- rte_rawdev_stop

- rte_ioat_enqueue_copy
- rte_ioat_perform_ops

- rte_ioat_completed_ops



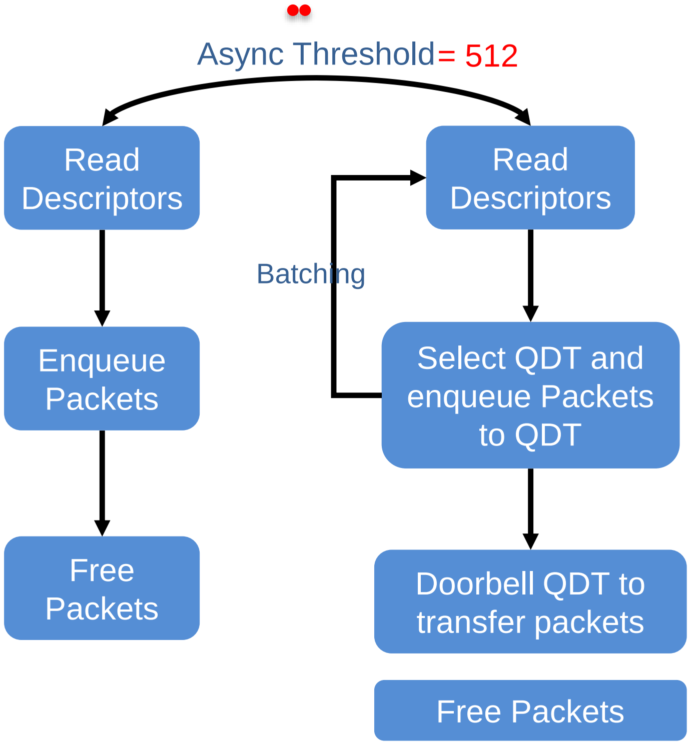
Note:

- **Only Enqueue operations supported currently**
- **All vHost async and IOAT API's are experimental**

Packet transfer pipeline:

CPU pipeline

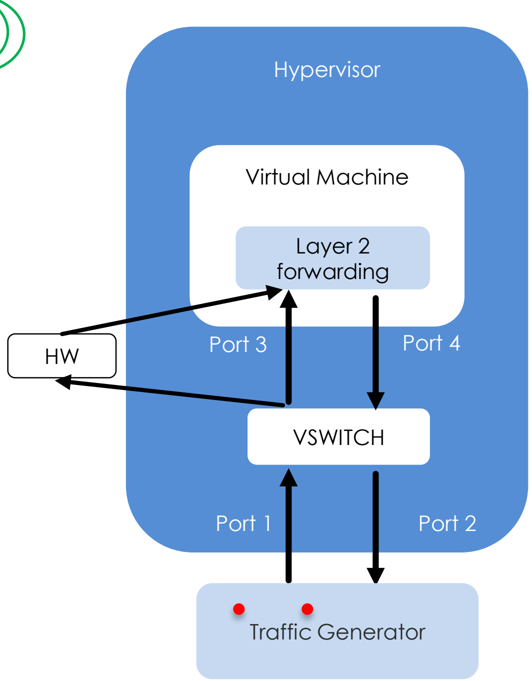
vHost async pipeline



Packet size = 256

Packet size = 1024

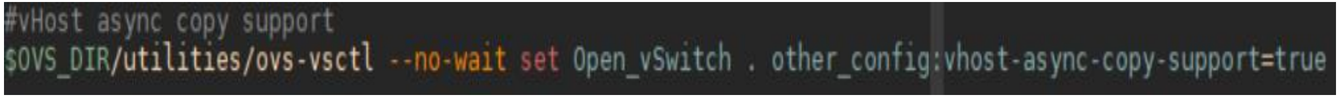
PVP



Note: Decision to choose the pipeline is made by the vhost library and not at application level.

How to enable vHost async ?

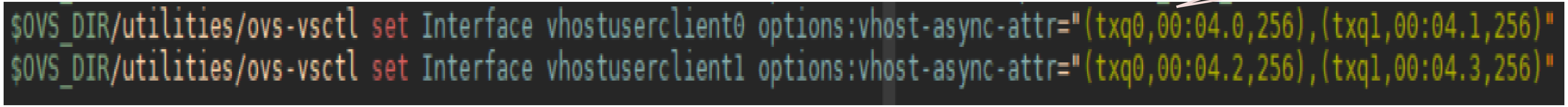
Enable async mode:



```
#vHost async copy support
$OVS_DIR/utilities/ovs-vsctl --no-wait set Open_vSwitch . other_config:vhost-async-copy-support=true
```

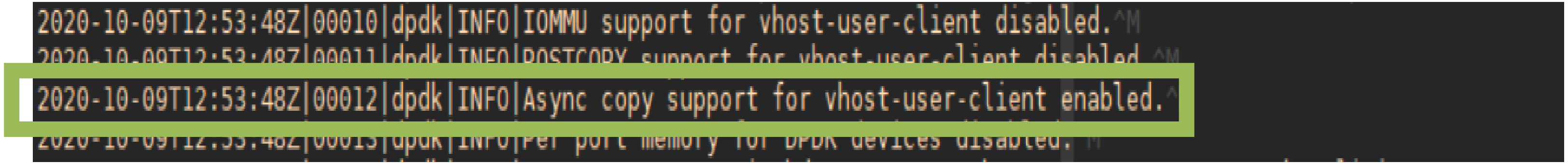
Set vhost async attributes:

txq#, DBDF, Async threshold



```
$OVS_DIR/utilities/ovs-vsctl set Interface vhostuserclient0 options:vhost-async-attr="(txq0,00:04.0,256),(txq1,00:04.1,256)"
$OVS_DIR/utilities/ovs-vsctl set Interface vhostuserclient1 options:vhost-async-attr="(txq0,00:04.2,256),(txq1,00:04.3,256)"
```

vswitchd.log:



```
2020-10-09T12:53:48Z|00010|dpdk|INFO|IOMMU support for vhost-user-client disabled.^M
2020-10-09T12:53:48Z|00011|dpdk|INFO|POSTCOPY support for vhost-user-client disabled.^M
2020-10-09T12:53:48Z|00012|dpdk|INFO|Async copy support for vhost-user-client enabled.^
2020-10-09T12:53:48Z|00013|dpdk|INFO|Per port memory for DPDK devices disabled.^M
```

Note: QDT channel must be bound to a user space driver like VFIO/IGB UIO.

Challenges:

1. rte_ioat_completed_ops returns number of “segments” sent while vHost library expects number of packets sent as a return.

2. QDT copy is asynchronous with CPU operations. QDT may still be copying packets when enqueue API returns. So, when to free and where? Also depends on HW.

3. QDT channel static mapped to Tx queues

4. Limited QDT channels

Possible solutions:

1. Have packet-segment tracking to match with rte_ioat_completed_ops return

2. a. Wait until all packets of current batch are free and then process next batch :

❑ wait in __netdev_dpdk_vhost_send

❑ CPU not doing any work other than waiting

❑ beats the purpose of async!

2. b. No wait, free inside the same function:

❑ Packets not free'd in current iteration/batch will be free'd next time.

❑ Have to call rte_vhost_poll_enqueue_completed to flush the virtqueue

❑ If dynamic txq , what if the same queue is not used to all from next iteration?

Possible solutions:

2. c. Free outside the send function much later:

Considerations:

❑ Needs to have access to the netdev and tx qid.

❑ Must be called regularly in the PMD.

Good contender:

❑ dp_netdev_pmd_flush_output_packets

❑ Called inside the PMD regularly

❑ Has access to netdev and tx_qid via pmd->send_port_cache

❑ Free once after dp_netdev_pmd_flush_output_on_port

❑ call free continuously for the netdev and qid when no packets to send between bursts.

But …

❑ Will require spinlock for txq

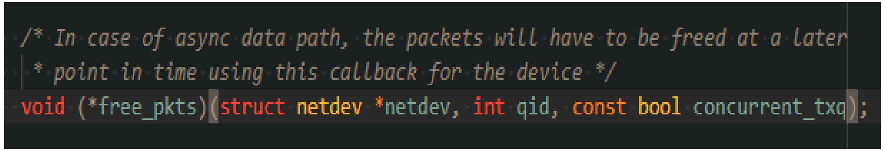
❑ High CPU usage of free function ~60%

❑ Breaks abstraction.

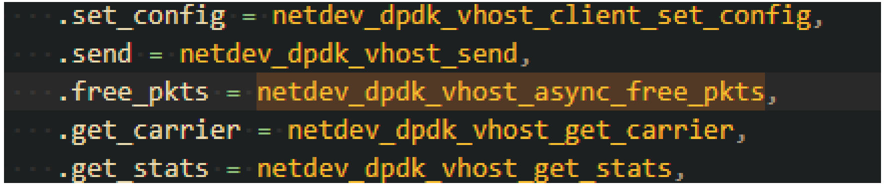
Possible solutions:

Reduce the CPU usage:

- ❑ Call free only for vhost ports by introducing callbacks at netdev level (struct netdev_class)



```
/* In case of async data path, the packets will have to be freed at a later
 * point in time using this callback for the device */
void (*free_pkts)(struct netdev *netdev, int qid, const bool concurrent_txq);
```



```
    .set_config = netdev_dpdk_vhost_client_set_config,
    .send = netdev_dpdk_vhost_send,
    .free_pkts = netdev_dpdk_vhost_async_free_pkts,
    .get_carrier = netdev_dpdk_vhost_get_carrier,
    .get_stats = netdev_dpdk_vhost_get_stats,
```

- ❑ Avoid calling free if no packets to free.
- ❑ Have tracking at netdev level perhaps a bitmask for each queue or may be even at pmd level ?
- ❑ CPU usage of free function ~ 7%
- ❑ What if dynamic txq again ?

Future Work

- Further investigation on where and when to free the packets
- Support vHost async dequeue operation.
- Support sharing QDT among vhost queues and ports.
- Introduce debuggability.
- Update documentation

Patch at :
https://patchwork.ozlabs.org/project/openvswitch/patch/20201023094845.35652-2-sunil.pai.g@intel.com/
Comments are welcome!

References

- https://doc.dpdk.org/guides-20.11/rawdevs/ioat.html
- https://www.intel.com/content/www/us/en/wireless-network/accel-technology.html
- https://www.dpdk.org/wp-content/uploads/sites/35/2019/10/Asynchronous.pdf
- https://www.dpdk.org/wp-content/uploads/sites/35/2018/12/JiayuHu_Accelerating_paravirtio_with_CBDMA.pdf

Thank You!

?Questions?

e-mail: sunil.pai.g@intel.com