Tuning the buffers: a practical guide to reduce or avoid packet loss in DPDK applications

V0.3

Contents

1 Summary

A network is designed with the right buffering in place at the right spots in the network. This is to avoid packets are lost in case there are not enough resources available for immediate processing in the next steps of a pipeline. When the buffer is not large enough, the packet loss will go up. When the buffers are too large, valuable resources are wasted and high latency might become an issue.

Finding the right balance between all available compute/network resources and the sizes of buffers is something we need to carefully design.

This balance is also important when looking at the interaction within a server between the network cards (which have some on-board buffering) and the DPDK managed buffer resources on the host. A better tuning of the buffer sizes can eliminate potential packet losses. This paper is summarizing what to do when going from one type of network card to another one that has different on-board buffer behavior. It also has the potential to explain and fix certain packet loss issues going from one generation of a NIC card to another (e.g. when moving from Intel® Ethernet Server Adapter X520 to Intel® Ethernet Controller XL710)

2 NIC buffer management

The management of buffers on a NIC can be implemented in many ways. Looking at it in a simplified way, NICs can have one pool of buffers shared between all available physical interfaces, or they can pre-allocate certain pools for one physical interface that cannot be used for other interfaces. Same is true for virtual interfaces exposed to the host via Virtual functions.

The buffers management system on the card can be implemented using a drop or no-drop scheme. This choice can be hard coded or available via a configuration parameter. A drop scheme means that the card will drop a packet immediately when the next step in the pipeline is not ready to accept the packet. This can happen when the NIC is trying to send a packet to the host which has no available host buffers or descriptors. A no-drop scheme means that the card will keep on trying to deliver the packet to the host till resources become available. This means the NIC is now at risk to run out of its own local buffers and packets will be dropped earlier in the pipeline.

The specific implementation of the on-board NIC buffer management is based on many considerations outside of the scope of this document. However, it is important to understand that each NIC comes with its own design choices that might affect the buffer management and its behavior: default drop vs no-drop, shared pools between multiple physical and/or virtual interfaces. It is not just about the buffer pool size.

3 Minimizing Packet loss

In a pipeline, a consumer processes packets provided by a producer. Both can have a buffer pool in case the consumer cannot deal with the traffic at this given moment in time. There are 2 reasons why packets cannot be processed in time by the consumer

- Traffic offered by the producer might be too high for the consumer to process given its compute resources.
- The consumer task gets interrupted by the OS and/or hypervisor

Both reasons are boiling down to the same basic problem: there are not enough CPU cycles available to deal with the incoming packets during the time window covered by the available buffers.

We now have to minimize packet-loss till a level that is acceptable for the function that this server is performing. In some cases, this means 0% packet loss needs to be achieved at the expense of applying more HW resources. There are 2 ways to minimize packet-loss.

- One way to minimize packet-loss is to avoid or minimize the interruption of the consumer task. This can be achieved using carrier-grade OS and hypervisors that are optimized for “real-time” behavior or by doing all these optimizations manually as described in chapter 5.

- The other way to minimize packet-loss is to optimize the buffers. How to do this is the subject of chapter 4.

4 DPDK and NIC configuration: optimizing the buffers

When using Niantic (PF), the packets can be either buffered in the RX descriptors (or, more correctly speaking, in the memory addressed by the RX descriptors), or in the RX packet buffer in the NIC. RX packet buffer size is 512KB when flow director is disabled, hence it can hold > 8000 packets of 64 Bytes (=> handle an interrupt longer than 500 microseconds).

> Calculation for 64 bytes: Each Ethernet packet has a 20 byte inter-frame gap and MAC preamble. So we need 84 bytes per packet. At 10Gb/s, this results in 14.88 M packets per second. To store all 14.88M packets of 64 bytes coming in during 1 second, we would need +- 908MB. This means we can buffer packets for 550 microseconds with the 512KB RX packet Buffer.

When using Fortville, packets are by default dropped when RX descriptors are not available. Hence, by default, the only buffers available to store packets when a core is interrupted are the buffers pointed to by the RX descriptors.

A similar behavior (packets dropped when no descriptors available) is obtained on Niantic when using Virtual functions: to avoid “head of line” blocking, packets are dropped if no descriptor is available and virtual functions are used.

So, to avoid packet losses due to CPU core being interrupted when using Fortville (or when using Niantic and SRIOV), the number of RX descriptors should be configured high enough, for instance to 2048.

- Setup number of rx descriptors through rte_eth_rx_queue_setup (see http://www.dpdk.org/browse/dpdk/tree/examples/l3fwd/main.c#n990)
  - ret = rte_eth_rx_queue_setup(portid, queueid, **nb_rxd**, socketid, NULL, pktmbuf_pool[socketid]);

The number of TX descriptors should also be considered and depends on the application. Imagine the CPU core is interrupted and up to 2048 packets are now buffered through the RX descriptors. Once the interrupt is dealt with, up to 2048 packets can be received very fast by the application: as the packets are already in memory, they might arrive much faster than 10Gbps. Hence, when the application tries to forward those packets, the application might try to send those packets faster than 10Gbps. Some of those packets must be buffered before being sent to the 10 Gb/s link. Either the application takes care of this by buffering those packets, or the packets must be buffered in the TX descriptors. In that case, the number of TX descriptors should also be increased (e.g. up to 2048).

Increasing the number of RX and/or TX descriptors implies that you might have to increase the number of pre-allocated mbufs. This can have a small impact on performance (throughput), as more memory is being used.

- Setup number of mbufs through rte_pktmbuf_pool_create: (see http://www.dpdk.org/browse/dpdk/tree/examples/l3fwd/main.c#n724)
    - rte_pktmbuf_pool_create(s, **nb_mbuf**, MEMPOOL_CACHE_SIZE, 0, RTE_MBUF_DEFAULT_BUF_SIZE, socketid);

5 Minimizing the interruption of the consumer task

Configuring the system to minimize interruption of the tasks dealing with the packets is key to achieve better performance, in terms of throughout, packet loss and packet delay variation.

Vendors of NFV solutions focus on getting these configurations right and hence it is a wise idea to check with your vendor if all these aspects are under control. Without trying to be a comprehensive list on what to do, one should look into:

- BIOS, hypervisor, host and guest kernel configurations
- Removing all services that are not needed (e.g. Bluetooth)
- Isolating CPUs that will be used for the fast datapath so they will not be interrupted by other tasks (see KVM4NFV, section 2.1.2)
- Optimizing key SW components e.g. vSwitch and using DPDK for the fast path applications.
- IRQ affinity pinning to housekeeping core (e.g. for the management network interface)