Veth XDP: XDP for Containers

Toshiaki Makita, *NTT* William Tu, *VMware Inc.*

Abstract

Veth native XDP support was introduced in kernel 4.19, which enables high performance networking in containers. It especially boosts network performance when used in conjunction with XDP_REDIRECT of physical network adapters. We show the design, usage, performance and challenges of veth native XDP support.

AF_XDP support is another work-in-progress challenge to make veth XDP more useful. A community project is ongoing to implement Open vSwitch datapath using AF_XDP, with which AF_XDP support in veth significantly improves performance of inter-container networking.

1. Introduction

XDP support for veth[1] is meant to boost veth networking performance, especially with redirection from physical network interfaces using XDP_REDIRECT action. This section describes the motivation and envisioned use cases of the feature.

1.1. Motivation

Veth is in many cases used for Linux containers. It acts as a mediator between containers and their hosts and transfers packets from each other.

These days containers are used in more variety of areas, such as Cloud native Network Function (CNF)[2] for NFV, in which network performance is critical. In order to achieve line-rate performance in containers using veth for their network adapters, we added XDP driver support for veth.

1.2. Use cases

Veth native XDP support can be used for several usages. Here we provide some examples we envision.

- **XDP in containers**

  As described in motivation section, the feature can be used to install XDP programs from containers. When containers are used as CNFs in NFV, the XDP programs will serve as network functions. Service function chaining is also doable using XDP_TX from the XDP programs on veth devices in containers in conjunction with XDP_REDIRECT on their peer devices.

- **XDP program chaining**

  It is also possible to realize program chaining by redirecting packets from one veth device to another veth device using XDP_REDIRECT.

2. Preliminaries

We explain existing features necessary to illustrate how native veth XDP support is different from existing solutions performance-wise.

2.1. XDP

XDP, eXpress Data Path[3], is a mechanism which implements fast data path for received packets in network drivers. XDP allows user applications to install eBPF programs, which can modify packets and determine how packets are treated at network driver level. Each driver calls this eBPF program immediately after it retrieves receive packet buffer, so there is minimum software overhead for packet processing, thus this mechanism is significantly faster than normal packet processing. However there are some limitations on XDP. One of them is that it requires each driver implementation by its nature, so users need to select devices whose drivers support XDP. Another limitation is that each driver has its own conditions to enable XDP: E.g. virtio_net requires twice the number of normal queues which can be configured only from hypervisors. This in turn requires users' deep knowledge about each drivers' requirements. XDP is called native XDP or driver XDP when compared with generic XDP described below.

2.2. Generic XDP

Generic XDP[4], introduced in kernel 4.12, is a fallback mechanism for driver level XDP implementation. As described in the previous section, XDP has some limitations. Generic XDP overcomes them at the expense of performance. It is implemented in network stack instead of each driver, thus with any device users can use it. It does not require any special setup before using it. The performance is lower than native XDP because it does not process packets immediately after received packets are retrieved and has software overhead. Generic XDP can be used for beginners' trial usage or testing.

Initial generic XDP implementation only allows usages with physical network interfaces. It is extended in kernel 4.14 and has been able to be used with virtual network interfaces including veth[5].

3. Veth native XDP support

As explained above, generic XDP is not optimal performance-wise, because it does not handle packets immediately after drivers retrieve received packets. With virtual network interface we do not have physical devices, thus we cannot handle packets immediately after drivers retrieve packets from hardware. However, if physical network interfaces pass raw packet buffers to virtual devices immediately after they retrieve packets, the XDP performance on virtual network interfaces can get close to driver XDP on physical interfaces, and can significantly improve the performance compared with generic XDP. In order to realize this mechanism, we can use XDP_REDIRECT action on physical devices. XDP_REDIRECT is an action meant to pass raw packet buffers retrieved by XDP to another devices.

We added veth native XDP support with this envisioned usage in mind. Note that handling packets sent from the peer's stack should not be better in performance than generic, because both approaches handle non-raw packet buffers with traditional sk_buff style packets thus driver XDP cannot benefit from low overhead of handling raw packet buffers.

3.1. Design

The veth driver originally used general softirq backlog (per cpu queue mainly used for hardirq to enqueue packets). Packets in this backlog are drained by the general softirq handler for received packets, process_backlog(). As this process is common for drivers without NAPI and RPS mechanism, we cannot insert a function to handle XDP packets.

Thus we added new dedicated NAPI handler for veth XDP, and receive queues for the handler in veth. With this approach, the tx side interface, the peer of the interface XDP is installed on, can enqueue raw XDP packet buffers redirected from physical interfaces to receive queues of the rx side interface and kick the NAPI. Then the NAPI handler drains receive queues and calls the installed XDP program. Using NAPI handlers also enables redirection of redirected packets, avoiding infinite loop and stack inflation. If we called XDP programs without enqueueing redirected packets, and the packets were redirected again so that it forms a redirection loop, it would never stop and stack would overflow.

3.2. Usage

- **Prerequisite**

  To install XDP programs, you need to become a root user. For the best performance, allocate the same number of queues as CPU cores. Also, turn off rx vlan offload on both veth and physical devices. Turn off tx checksum offload on veth devices. Note that we need to disable tx vlan offload of the peer device to disable rx vlan offload of a veth device.

- **Without XDP_REDIRECT**

  For this usage you only need to install an XDP program on a veth device. Note that this does not boost performance as described before.

- **With XDP_REDIRECT of physical interface**

  This requires an XDP program installed on the peer device of the target device of redirection, e.g., if you have a veth pair whose devices are veth0 and veth1, and if you redirect packets from a physical interface to veth0, you need to install an XDP program on veth1 to make packets reach veth1. Otherwise packets redirected from the physical interface to veth0 will be dropped because receive queues on veth1 is not ready when XDP programs are not installed.

3.3. Performance numbers

We measured performance with XDP_REDIRECT of physical interfaces comparing with generic XDP. We used 2 machines connected through a switch. Other information on the machines are as follows:

- CPU: Intel Xeon Silver 4114, 2.20GHz, 2-socket, 10-core each.
- NIC: Intel XXV710 25-Gigabit, i40e driver.
- Kernel: 4.20.13

Note that spectre_v2 mitigation is disabled on the machines.

We ran 3 types of tests as follows:

- **XDP_DROP test**

  Redirect packets from the physical interface to a veth device, veth0, and drop them by XDP on its peer veth device, veth1.

- **XDP_TX test**

  Redirect packets from the physical interface to a veth device, veth0, send them back by XDP_TX action on its peer veth device, veth1, and drop them on veth0.

- **XDP_REDIRECT test**

  Redirect packets from the physical interface to a veth device, veth0, redirect them again by XDP_REDIRECT action on its peer veth device, veth1, to another veth device, veth2, and drop them on its peer veth device, veth3.

In all tests we used pktgen to generate traffic from the sender machine. The sending rate is approximately 37 Mpps, which is almost line-rate of 25 Gbit Ethernet. We measured the number of packets counted in each XDP program which drops packets. The tests are done with 1-flow and 100-flow, which exercises 1-core and 20-core respectively. The results are shown in figure 1 and figure 2.

Figure 1: 1-flow (1-core) performance tests results

1-flow (1-core) tests show that native XDP achieves more than 10 Mpps on XDP_DROP test, approximately 7 Mpps on XDP_TX and XDP_REDIRECT tests, while generic XDP was approximately 2 Mpps on each test.

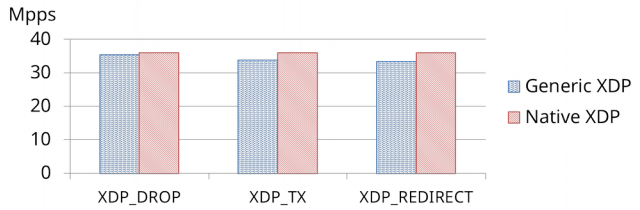

Figure 2: 100-flow (20-core) performance tests results

100-flow (20-core) tests show that both type of XDP scale by cores, but native XDP has slightly better performance than generic.

3.4. Challenges

- **XDP_TX performance**

  With current implementation, XDP_TX of veth acquires the receive queue lock per packet. This is not efficient compared with XDP_REDIRECT which acquires the lock once in a bulk sending. We should implement bulk sending for XDP_TX as well to improve XDP_TX performance.

- **Intuitive way to enable XDP_REDIRECT**

  Redirection to veth devices requires installing XDP programs on their peer veth devices. This is not intuitive and users may spend a lot of time to make it work correctly. We need to find out a more intuitive way to enable redirection.

- **Virtual switch**

  As veth XDP requires XDP_REDIRECT for the best performance, we need an XDP-based virtual switch.

4. Veth AF_XDP

4.1. AF_XDP Basics

AF_XDP[9] is a technology built on top of eBPF and XDP. By leveraging XDP, the AF_XDP executes in the context of the NIC's driver, which runs before serveral memory allocation and other subsystems. An XDP program can decide which packets should continue processing in the kernel and which should be redirected to another interface or dropped. An AF_XDP defines a socket interface that allows userspace programs to receive and send raw frame from the XDP program specifying a redirect action to the AF_XDP socket. Due to the nature of XDP running in the context of driver level and several optimizations, there is minimal amount of overhead for AF_XDP socket to access the raw frame.

4.2. Veth AF_XDP and OVS use case

One use case of AF_XDP support in veth is for OVS to receive, process, and forward packets between different network interfaces. We have been working on AF_XDP supports for OVS[7][8], and currently the performance is good only when the network interfaces are i40e or ixgbe, which has AF_XDP driver support.

When OVS tries to forward a packet to a non-AF_XDP supported driver, OVS has to re-inject the packet into the kernel and as a result, causing lots of context switches and copying overhead. To be able to get similar performance when traffic traverses between VMs and containers, this requires AF_XDP supports for virtual devices such as tap and veth drivers.

Currently, the ixgbe/i40e driver with AF_XDP support can copy packet buffer directly to and from the NIC's DMA engine, an execution mode called zero-copy mode. For a veth device, since there is no underlying physical device, the design is different from the physical NIC driver. As a result, we have been working and discussing about the implementation and an RFC implementation of AF_XDP veth driver can be found[6].

4. Conclusion

We added veth native XDP support in kernel 4.19. This improves XDP performance in containers from approximately

2 Mpps to 10 Mpps, comparing with generic XDP. This can be used for service function chaining in NFV.

AF_XDP support for veth is work in progress. This can be used for AF_XDP datapath of Open vSwitch to improve inter-container network performance.

5. References

[1] Toshiaki Makita, “veth: Driver XDP”, commit 60afdf066a35, August 10, 2018, https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=60afdf066a35

[2] CNCF, “Cloud-native Network Function (CNF) Testbed”, GitHub website, https://github.com/cncf/cnf-testbed

[3] IO Visor Project, “XDP eXpress Data Path”, IO Visor website, https://www.iovisor.org/technology/xdp

[4] David S. Miller, “net: Generic XDP”, commit b5cdae3291f7, April 25, 2017, https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=b5cdae3291f7

[5] John Fastabend, “net: xdp: support xdp generic on virtual devices”, commit d445516966dc, July 17, 2017, https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=d445516966dc

[6] William Tu, “AF_XDP support for veth”, December 26, 2018, https://www.spinics.net/lists/netdev/msg542047.html

[7] William Tu, “AF_XDP netdev support for OVS”, November 28, 2018, https://patchwork.ozlabs.org/cover/1004837/

[8] William Tu, “Fast Userspace OVS with AF_XDP”, *Open vSwitch 2018 Fall Conference talk session slides*, 2018, https://ovsfall2018.sched.com/event/IO7p/fast-userspace-ovs-with-afxdp

[9] Björn Töpel, “Introducing AF_XDP support”, January 31, 2018, https://lwn.net/Articles/745934/