Using SR-IOV offloads with Open-vSwitch and similar applications

Rony Efraim, Or Gerlitz
Mellanox
Ra'anana, Israel
ronye@mellanox.com ogerlitz@mellanox.com

Abstract

Cloud use cases such as NFV pose high performance requirements (both in terms of bandwidth and real-time-ness) on hypervisors. When all VM traffic is handled by a host switching entity (such as when emulated or Para-Virtual guest network interfaces are used), a very high load on the local host CPU is implied. Thus, the traditional way of providing network access to VMs doesn't meet the performance requirements. SR-IOV devices present opportunity to improve network performance. Up until recent versions of the Linux kernel, SRIOV environment posed deep limitations on the ability of the hypervisor to manage the network when flow based approaches like Open vSwitch and TC or IP based tunnels were used. Recently, an approach that facilitates SR-IOV performance while maintaining flow-based management using the TC subsystem framework to program the SR-IOV e-switch was introduced in the Linux kernel. This allows to have hardware offloads for most traffic, and a software fallback for control traffic. We show how to use the new infrastructure in the kernel from the OVS daemon (or similar software switching applications) for achieving HW offloaded data-path in SRIOV environment.

Keywords

Virtualization, SR-IOV, switchdev, TC, Flower, OVS, offload, flows, learning, tunnels

Introduction

In a virtual server environments, the most common way to provide Virtual Machine (VM) switching connectivity is via a Virtual Switch sitting in the hyper visor. The Virtual switch is basically a software switch that acts similar to a Layer 2 hardware switch providing inbound/outbound and inter-VM communication.

One of the dominant virtual switches is OVS (Open Virtual Switch) which switches frames between local VMs on the host (sometimes called east-west traffic) and between local VMs and remote VMs (sometimes called north-south traffic). One major difference between OVS and a "regular" IEEE Ethernet bridge is that OVS switches "flows" as oppose to a regular Ethernet bridge which provides frame delivery between VMs based on MAC/VLAN.

Traditional hypervisors expose emulated or para-virtual (PV) devices to guest virtual machines and multiplex the I/O requests of the guests onto the real hardware through the software switch residing in the host OS. Each emulated or PV device assigned to a guest has a TAP device associated with it on the hypervisor where this interface is plugged to the virtual switch along with the uplink NIC port.

Emerging technologies such as NFV and other performance-sensitive cloud use cases, call for high performance requirements (both in terms of bandwidth and real-time-ness) on data plane devices within network infrastructure. Since virtually all of the VMs traffic is first handled by the host switching entity, this implies a very high load on the local host CPU implementing the virtual switch. Hence, the traditional way of providing network access to VMs doesn't meet the performance requirements.

SR-IOV is a specification by PCI-SIG that allows a single physical device to expose multiple virtual devices. Those virtual devices can be safely assigned to guest virtual machine giving them direct access to the hardware. Using hardware directly reduces the CPU load on the hypervisor and usually results in better performance and lower latency.

The way SR-IOV embedded switches are dealt with in Linux is limited in its expressiveness and flexibility, but this is not necessarily due to hardware limitations. The kernel software model for controlling the SR-IOV switch simply did not allow the configuration of anything more complex than MAC/VLAN based forwarding. Hence the benefits brought by SRIOV come at a price of management flexibility, when compared to software virtual switches which are used in Para-Virtual (PV) schemes and allow implementing complex policies and virtual topologies.

In our work we have created a change towards getting the best of both worlds: the performance of SR-IOV with the management flexibility of software switches. This provides a richer model for controlling the SR-IOV e-switch for flow-based switching and tunneling. The code we've integrated into the upstream Linux kernel provides the infrastructure for using virtual switches such as OVS in SRIOV environment.

Relation to previous work

The problem of HW offloading flow based SRIOV datapath has been previously discussed in [1]. The implementation which is described in [1] took an approach of directly offloading the flow rules from the Open-vSwitch (OVS) kernel software datapath through an extended kernel switchdev API exposed by VF representor net-devices.

The solution described in the following sections which is now merged into the upstream Linux kernel is slightly different. It does involve VF representor net-devices as in the earlier work. However, the offloading decision is done in user-space and is programmed to the HW driver through the TC subsystem. This difference provides a more generic mechanism that supports multiple SW switching implementations, not limited to OVS. It also allows for integration with user-space software policy engine for making the offloading decision.

SRIOV switchdev offloads mode, VF Representor net-devices

Up until recently, the way SR-IOV embedded switches were dealt with in Linux was limited in its expressiveness and flexibility. The kernel software model for controlling the SR-IOV switch did not allow the configuration of anything more complex than MAC/VLAN based forwarding. We refer to this model as the legacy mode.

With kernel version 4.8, a new mode (switchdev) for SR-IOV switches was introduced. Similarly to the switchdev drivers serving HW switch ASICs, a SW representation model is used (here for the SR-IOV E-Switch ports) which in turn allows to offload the SW switch traffic rules to the HW e-switch.

The new model is based on the introduction of per VF representor host net-device. The VF representor plays the same role as TAP devices in Para-Virtual (PV) setup. A packet sent through the VF representor on the host arrives to the VF, and a packet sent through the VF is received by its representor. The administrator can hook the representor net-device into a kernel switching component. Once they do that, packets from the VF are subject to steering (matching and actions) of that software component.

Doing so indeed hurts the performance benefits of SRIOV as it forces all the traffic to go through the hypervisor. However, this SW representation is what eventually allowed to introduce hybrid model, where some of the VF/VM traffic is offloaded to the HW while keeping other VM traffic to go through the hypervisor. Examples for the latter are first packet of flows which are needed for SW switches learning and/or matching against policy database or types of traffic for which offloading is not desired or not supported by current e-switch HW.

Basic setup of NIC Embedded switches is expected to be through a PCI device driver. As such, a devlink/pci based scheme for setting the mode of the e-switch was introduced.

The first upstream driver to implement the new SRIOV switchdev scheme is mlx5 which is serving the Mellanox ConnectX4 NIC ASIC family. In the mlx5 driver, the VF representors implement a functional subset of mlx5 Ethernet net-devices using their own profile. This design bought us robust implementation with code reuse and sharing [8].

The representors are created by the host PCI driver when set in SRIOV and the e-switch is configured to switchdev mode. Currently, in mlx5 the e-switch management is done through the PF e-switch vport and hence the VF representors along with the existing PF net-device which represents the uplink share the PCI PF device instance.

In mlx5, the initial setup of the e-switch to serve for software based switching is provisioned with the following:

- A single miss rule which matches all packets that do not match any other HW switching rule; and the action is to forwards packets to the e-switch management port for further processing. Eventually these packets show up at the representation port. Since all the VF reps run over the same e-switch port, we use more logic in the host PCI driver to do HW steering of missed packets into the HW queue opened by a the respective VF vport representor.

- send-to-vport rules which are higher priority than other "normal" steering rules which present at the e-switch datapath. Such rules apply only for packets that originate in the e-switch manager vport.

Flow API using TC, TC HW offloads

In the recent years, few APIs were suggested in the Linux networking kernel community to support flow programming from user-space. The term flow rule refers to certain set of matchings on packet headers; an associated set of actions is typically applied on the matching packet. a match. The TC architecture is described in [10].

Two reccent proposals are: John Fastabend's Flow API [2] and Jiri Pirko's TC *Flower* classifier [3][4], with the latter being upstream since version 4.2 of the kernel. *Flower* allows to match on L2/L3/L4 packet headers with wild cards (masking) as typically required in open-flow based environments where the matching part is stated in a manner which is common for ACLs/TCAMs.

Another building block towards being able to actually offload flow rules to HW was the introduction of a framework for offloading TC classifier rules/actions to NIC HW drivers. This was done through Amir Vadai's [5] and John Fastabend's [6] work to extend the *setup_tc* NDO (net-device-operation) which was previously used only for setting HW QoS, this extension was merged into version 4.6 of the upstream kernel. John enabled the TC *U32* classifier offloads onto the Intel 10Gbs ixgbe NIC driver and Amir enabled TC Flower classifier offloading in the Mellanox 100Gbs mlx5 NIC driver. The initial offloaded actions were HW *drop* (turn out that for both humans and HWs, the easiest thing is to say "no"…) in both drivers and in mlx5 also packet marking based on HW matching (referred to as skbedit in TC speak). HW based marking is a means to reduce CPU utilization in Para-Virt based SW switching datapath, using a breakdown of the rule to two (1) HW: original matching → mark and (2) SW: mark → original action.

Since that, more classifier offloads have been accepted or are proposed at time of writing. One of them is the *matchall* classifier by Yotam Gigi and Jiri Pirko which is used for offloading port mirroring in the mlxsw switchdev driver which serves the Mellanox 100Gbs switch ASIC HW. More proposals deal with TC actions to handle IP tunneling, offloading e-BPF classifier, actions to implement sFlow on switches, etc.

The kernel TC offloading framework supports rule addition, replacement, deletion and statistics, where the offloaded rule matching part is stated in a manner which is special to the classifier (e.g *Flower*, *U32*, *eBPF*) and the actions part are provided in their generic form.

Another element which is critical when offloading SW based switching to NIC and Switch HWs is the ability to specify for a given flow if it should be programmed to the SW datapath, the HW datapath or both. In TC, setting a rule only for HW is referred to as *skip-sw*, only for SW is referred to as s*kip-hw* and if none of them is specified, the default behavior is to put the rule in the SW datapath and in parallel to the HW one. Under this default, only if setting to SW failed, the rule addition operation fails. This means that HW offload under the default case is done in best effort manner. The UAPI (API towards user-space) for TC HW offloading (e.g the offload policy for a certain rule) was embedded into the TC UAPI in version 4.6.

TC based HW e-switch datapath

The case of offloading switching rules between different HW ports such as NIC SRIOV E-Switch ports or Switch ASIC ports in nicely served by the TC HW offloading framework. This is due to the fact that in Linux switch/e-switch ports are modeled as net devices and when this port net-device supports TC offloads, switching rules can be stated as ingress TC classifier rules programmed to the port (switch ASIC port or SRIOV VF vport representor) net-device.

Common examples for matching required for switching offloads are link layer (Ethernet and VLANs) headers and in some environments also network layer (IPv4/IPv6) headers or transport layer (TCP/UDP) headers. Common examples for actions required for switching offloads are *drop* and *forward*, potentially with doing *push* or *pop vlan* headers. The TC action used for forwarding is called *mirred* with a sub-action called *redirect* (*mirred* is also used for port/packet mirroring).

When forwarding rule is offloaded to the HW, one must be sure that both ports (ingress and egress) belong to the same ASIC. This is achieved by looking on the switchdev HW parent ID attribute which is unique per ASIC ID. Only if the IDs match HW offloading is possible. Another matter to be taken care of is flow statistics and aging. Packet/byte statistics are used by management and monitoring systems for various needs. SW switching typically does aging by apply SW polling strategy over a per flow "last used" time-stamp maintained by the SW datapath. The kernel TC HW offloading framework does allow HW port driver to report statistics and last-use time stamp for offloaded flows. In case the HW doesn't support the last-use time stamp, some emulation can be done by running a background thread that polls the HW flow statistics every once in a while and caches them, where the time-stamp of a flow is incremented only when the packets counter of the current sample is bigger from the previous one.

In the SRIOV E-switch case, we achieved TC based HW datapath through TC offloads by the mlx5 VF representor netdevices implementing the *setup_tc* offload ndo. This was also merged in version 4.8, where the supported elements are L2/L3/L4 matching along with the *drop* or *forward* (*mirred redirect*) actions [9]. TC vlan action offloading by the mlx5 VF representors accepted upstream for version 4.9.

Handling IP tunnels with TC

The kernel networking subsystem has rich support for IP tunnels. Two commonly used IP tunnels are VXLAN and NVGRE (GRE with inner protocol being TEB) where the implementation of both is based on setup of shared tunneling net-device. When these devices are used, encapsulation is done by a higher level element in the networking stack (such as SW switch datapath) attaching the tunnel meta-data (MD), which is the IP/UDP information needed in order to construct the encapsulation headers to the kernel packet structure (skb) and invoking the xmit NDO of the tunnel net-device. The way decapsulation works in this setup, is by the tunnel net-device striping off the tunnel info from the packet buffer and attaching it as meta-data to the packet skb. Later this MD can be used by these higher level elements in the stack for matching and switching the packet e.g. to VMs in para-virtual environment.

The approach taken in the TC implementation was to enable TC based SW datapath which use such tunnel devices. A work by Hadar Hen-Zion and Amir Vadai [7] introduced two new TC actions “tunnel key set” and “tunnel key release”, they also extended the Flower classifier to match on the tunnel info, this has been accepted upstream and to be merged into version 4.9-rc1 of the kernel. For encapsulation, the user programs a TC ingress rule on the original end-point, e.g. on a VM host Para-Virtual tap device or SRIOV VF vport representor net-device. The rule does a match on the original packet along with tunnel key set action which gets the tunnel info and attaches it to the skb as meta-data, followed by a mirred/redirect action which transmits the packet to a tunneling device. For decapsulation, the user programs a TC ingress rule on the tunnel device. The rule does a match on both outer and inner parts of the packet along with tunnel key release action which simply releases the meta-data, followed by a mirred/redirect action which transmits the packet to the actual end-point (e.g VM host tap interface).

TC based HW e-switch datapath for IP tunnels

Before attempting to deal with IP tunnels, TC E-Switch HW offloading is done for rules describing packets received to the HW datapath through a HW port, where some matching is required, followed by local actions on the packet (dropping, marking or adding/removing vlan tag) and finally forwarding the packet to an egress port. All to all, a combination of HW source port, matching, actions and HW destination port is offloaded from the TC subsystem to the e-switch through the port NIC driver.

With IP tunnels, the TC rules involve the SW tunnel net-device which doesn’t have direct access to the port HW. There are multiple ways to address this challenge, as was discussed in [1]. The strategy which is now in the works by the authors of the TC tunnel key set/release actions, is to use route lookups for realizing what the HW ingress device for the decapsulation rule is and egress device for the encapsulation rule. Another challenge which is also discussed in that paper is L2 neighbour resolution and maintenance for the tunneled packets.

Building a software switch using HW offload infrastructure

Combining the infrastructure elements presented in the previous sections, we now show how to build a software switch what utilizes the HW offloads components – SRIOV VF representors and TC offload APIs both implemented by a HW e-switch device driver.

Use the software datapath (kernel) as a slow path

By default a miss rule is installed in the HW. Packets not belonging to any offloaded flow hit the miss rule and are delivered by the HW to the e-switch management host port along with meta-data stating from what HW vport they were received. This is used by the e-switch driver to get the missed packet be received into the host OS from the representor of that port (VF or uplink).

The packet will show up in the software data-path and would experience a miss there too. Once this happens, the packet is sent to the switch management software. If the decision is to offload the flow, a TC call will be made to configure that into the HW through the respective representor. The original packet will be re-injected to the e-switch on the egress port representor which is dictated from the new flow and hence will show up in the correct destination.

Flow aging & counters

Data driven switch population rules are based on traffic. The switch software adds a rule when packet arrives and removes it when there is no traffic that matches that rule. Since the HW offload scheme prevents the software data-path from receiving and processing packets belonging to the offloaded flows, the per flow software counters will not show any progress. As a result the aging mechanism will age out those flows. In order to avoid this, the switch software needs to be augmented with a mechanism that reads counters of flows that are offloaded to the embedded HW switch.
Same as for adding/deleting rules, the mechanism to read the traffic counters and last used time of flow rules is TC.

The switch software polls the counters and last used of all the rules through TC and removes the rules that were not been used for long time.

Policy considerations

HW offload should support the ability for the system administrator (as well as the control plane itself) to dictate which flows get offloaded and which not. This is needed for various reasons, among them are pure-policy (i.e. to differentiate between customers/VMs/etc.), complex processing of flows that require CPU handling (e.g. flows that requires statefull inspection/DPI) etc. To achieve that, some policy module is needed. That module receives policy definitions from the user. It deploys some logic derived from the policy and from the various processing procedures that a flow needs to undergo, and orders which flows get offloaded and which not.

Working example of HW offloads with Open-VSwitch (OVS)

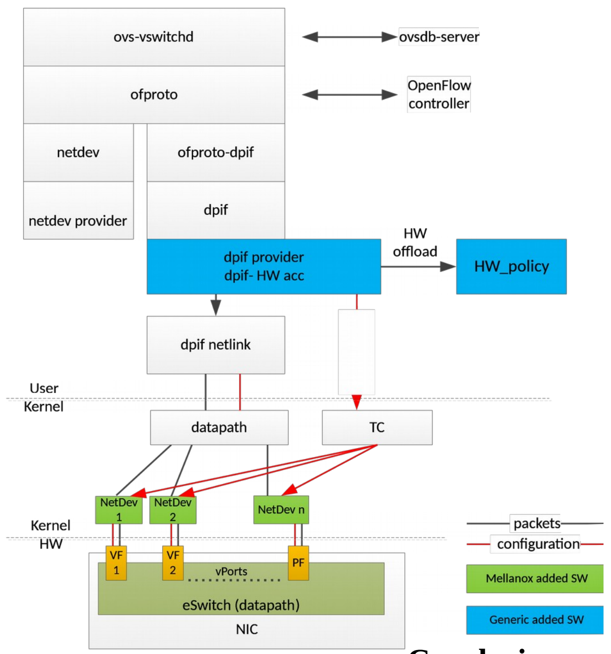
One of the dominant virtual switches is OVS (Open Virtual Switch) which switches frames between local VMs on the host and between local VMs and remote VMs. A major difference between OVS and a "regular" IEEE 802.3 Ethernet bridge is that OVS switches "flows" as oppose to a regular Ethernet bridge which provides frame delivery between VMs based on MAC addresses and VLAN tags.

Legacy OVS works in a "fastpath-slowpath" concept. The dataplane is used as a cache for the decisions made by the control plane. In this context the OVS kernel module implementing the datapath is the fastpath and the ovs-vswitchd implementing the control plane is the slowpath.
Upon reception of a packet of a flow "unknown" to the dataplane (i.e. there is no matching forwarding entry in the cache) the packet is forwarded to the user space daemon (ovs-vswitchd) to be further processed. The ovs-vswitchd daemon consults with its own database (which can be updated/modified by various SW entities, resulting in the "SDN" nature of OVS) and then installs a forwarding entry (a.k.a a flow rule) in the kernel datapath module. Thus, subsequent packets will be handled by the kernel.

In order to preserve the SDN nature of the OVS and to maintain the above model, we will retain both the control plane and the dataplane entities, and will augment them.
We build an openVswitch (OVS) switch with the VF representors and the uplink representor (PF), where the default is that any missed traffic will be received by the respective representor.

The kernel datapath of OVS will forward the received packet to ovs-vswitchd that will process it and generate a flow rule to handle this traffic. If ovs-vswitchd decide to HW offload this flow (based on the policy) it programs a HW only ("skip sw") TC rule with the required classification and action. Future packets with this classification will hit the e-switch rule and will be forwarded in HW. The ovs-vswitchd aging of old flows is done by polling all the rules every few seconds. The offloaded HW rules are polled too through TC and removed according to aging policy of OVS. Packets forwarded by the kernel datapath are transmitted on the representors and forwarded by the e-switch to the respective VF or to the wire. In order to support HW policy in OVS and to use TC it is required to have a new dpif provider which we named dpif-HW-acc which according to policy either uses TC or dpif netlink.

This way the OVS code does not "change"… we plug our new dpif HW acceleration module that exposes the same APIs that a dpif provider should expose and use the standard dpif netlink. We keep all the changes within the dpif hw acceleration module. The new module calls the HW policy module per flow add/remove to realize where this should be offloaded or not. According to the reply it calls dpif-netlink if it is not offloaded or calls TC if it should be offloaded. On flow dump OVS dumps flows from the dpif-netlink and then also from TC.



Conclusion

We have implemented and upstreamed a design that allows to enjoy the performance advantages of SRIOV while keeping the flow based approach to guest traffic used in Open-vSwitch. The building blocks we have added to the kernel can be used for similar (non OVS) hypervisor software switching systems as well. Support for offloading IP tunnels in this framework is in the works and once it meet upstream, the overall solution could be applied on environments that require that.

Acknowledgements

We'd like to thank Haggai Eran and Ilya Lesokhin for their deep technical contribution to the earlier version and during the upstreaming of the approach presented here. Thanks a lot to our team members Hadar Hen-Zion and Amir Vadai for their professionalism, dedication and patience on the upstream trail. A lot of coaching and guidance we're getting from the Mellanox switchdev team folks, Ido Schimmel and Jiri Pirko, thanks for your firm leadership and steady humor.

References

[1] Lesokhin I, Eran H, Gerlitz O - "Flow-based tunneling for SR-IOV using switchdev API", netdev 1.1 http://www.netdevconf.org/1.1/proceedings/papers/Flow-based-tunneling-for-SR-IOV-using-switchdev-API.pdf

[2] Fastabend, J. 2015. A flow api for linux hardware devices. http://people.netfilter.org/pablo/netdev0.1/papers/A-Flow-API-for-Linux-Hardware-Devices.pdf

[3] Pirko, J. 2015a. Hardware switches - the open-source approach. http://people.netfilter.org/pablo/netdev0.1/papers/Hardware-switches-the-open-source-approach.pdf

[4] Pirko, J. 2015b. Implementing open vswitch datapath using tc. http://people.netfilter.org/pablo/netdev0.1/papers/Implementing-Open-vSwitch-datapath-using-TC.pdf

[5] Vadai A. commit 5b33f48 "net/flower: Introduce hardware offload support" http://git.kernel.org/cgit/linux/kernel/git/davem/net.git/commit/?id=5b33f48842-fa1e13e9c0ea8cc59c1d0df19042db

[6] Fastabend J. commit a1b7c5f "net: sched: add cls_u32 offload hooks for netdevs" http://git.kernel.org/cgit/linux/kernel/git/davem/net.git/commit/?id=a1b7c5fd7fe98f51fbbc393ee1fc4c1cdb2f011

[7] Vadai A. Hen-Zion H. commit d0f6dd8 "net/sched: Introduce act_tunnel_key" http://git.kernel.org/cgit/linux/kernel/git/davem/net-ext.git/commit/?id=d0f6dd8a914f42c6f1a3a8c08caa16559d3d9a1b

[8] Hen-Zion H. Gerlitz O. commit 513334e18 "merge branch mlx5-next" http://git.kernel.org/cgit/linux/kernel/git/davem/net.git/commit/?id=513334e18a74f70c0be58c2eb73af1715325b870

[9] Gerlitz O. commit 53d94892e "Merge branch mlx5-bulk-flow-stats-sriov-tc-offloads" http://git.kernel.org/cgit/linux/kernel/git/davem/net.git/commit/?id=53d94892e27409bb2b48140207c0273b2ba65f61

[10] Hadi-Salim J. Linux Traffic Control Classifier-Action Subsystem Architecture

http://www.netdevconf.org/0.1/proceedings/papers/Linux-Traffic-Control-Classifier-Action-Subsystem-Architecture.pdf

Authors Biographies

Or Gerlitz is a Linux kernel developer dealing with networking, RDMA and storage. He is an active contributor to the Mellanox upstream NIC drivers (mlx5 and mlx4).

Rony Efraim is software architect dealing with software interfaces to the Mellanox NIC ASIC product line.