Devlink enhancements for subfunctions management

Parav Pandit
Nvidia

Austin, USA

parav@nvidia.com

Abstract—PCIe SR-IOV is well known standard for accelerating I/O virtualization. Combining SR-IOV with Linux tc offloads achieves best of performance, cpu efficiency and network isolation. However newer deployment modes require high density virtual functions with faster life cycle than what PCIe SR-IOV can achieve. We discuss use cases which require scalable and high-performance virtual devices for bare-metal, containers, virtio acceleration and VM live migration use cases.

This paper discusses how Linux devlink subsystem is used to life cycle, configure and deploy accelerated sub functions with eswitch offloads support. Next, we will discuss the plumbing done using virtbus to achieve persistence naming of netdevices and rdma devices. We will also cover how such model addresses smartnic use case where a sub-function NIC is hot plugged in host system in secure manner.

Keywords—PCIe, SRIOV, subfunction, smartnic, virtual NIC, vnic, hot plug, virtio, Container, Docker, Kubernetes.

I. Introduction

PCIe SR-IOV is well known and widely used standard for accelerating IO-virtualization. In two major use cases PCIe SRIOV based network devices are used with (a) containers and (b) VM. When used with container, each container instance gets its dedicated SRIOV VF device and its associated network and rdma device [9]. SR-IOV networking devices also have mature offload interface using eswitch switchdev mode through which several offloads capabilities are used. Such offloads features enable users to provide network isolation, police and QoS configuration. This also enables user to deploy multi-tenant applications on a system while still maintaining high level of performance. However, SR-IOV is relatively requires higher system and device resources. It also hits scalability limitation 256 or 512 virtual functions per PCI device. Such scalability limitations arise from its deployment time needed to use a PCI VF, required PCI BAR and MSI-X table resources. In this paper we propose to overcome these limitations using subfunctions technology not only addressing scalability and performance requirements, but it also enables smartnic based bare metal deployment use. This paper proposes the devlink plumbing along with virtbus interface to enable user to achieve required scale, performance and various user requirements. This paper further describes the design of the detailed RFC submitted at [1] and [2] to linux kernel community.

A. Abbreviations and Acronyms

CNI: Container Networking Interface

PCIe: PCI Express

RDMA: Remote direct memory access

SF: Subfunction

SR-IOV: Single root input output virtualization

VM: Virtual machine

VF: Virtual function

II. User Requirements

This section describes various user requirements and use cases which drive the user interface and required kernel plumbing to achieve the same.

a. Ability to create, configure, deploy virtual device from a NIC side
When a user is using a smartnic device in a bare metal server, a eswitch is located on the smartnic device. On user's request through the rich cloud orchestration, system administrator or a cloud service

provider should be able to hot plug a new NIC device from the NIC system.

Often NIC management was limited to NC-SI or another proprietary interface in past. With the advent of smartnic devices, NIC management interface is more easily usable in open source manner through a standard linux software stack.

b. Same user interface from NIC side or the host side

   Not all use cases involved smartnic. A container use case without a smartnic should still be able to create such virtual NIC interfaces. It is desired to have single user interface regardless of deploying such virtualized NIC via smartnic or otherwise. This enables leading orchestration software such as Kubernetes, Kubernetes device plugin and CNI to deploy virtualized NIC using single programming interface.

c. Able to configure device before its discovered by OS and/or pre boot drivers

   In bare-metal use case cloud service provider should be able to fully configure the device before such device is plugged into the host. Such configuration includes but not limited to overlay network configuration, QoS, policy, MAC address. Performing hot plug of device without necessary configuration, and just by keeping the link down of the interface doesn't address all the requirements. A pre-boot environment with very limited memory and compute resources, typically runs in polling mode and not resilient to changes in network device attributes. Hence it is desired to perform necessary device configuration before inserting such device in the host or guest system.

d. Single PCI (PF/VF/SF) device supporting multiple class of devices at a time such netdev, rdma, vdpa and more

   A device may support multiple classes of device. These classes have well defined definition in Linux kernel. These classes include netdevice, rdma (infiniband), vdpa and more.

   A user creating only a Ethernet NIC device may not suffice the need to create such multiple classes of device. Hence a more generic user interface is needed to create the bus level subfunction device.

e. Ability to share low level resources with parent PCI device such as IRQ vectors, BAR region and more

   Unlike PCIe SR-IOV subfunction devices should be able to share the resources with its parent PCI device. In scenarios where device level isolation is needed, a device must be capable enough to have dedicated resource(s) per device. To begin with, when a device is used for RDMA applications, portion of the PCI BAR must be dedicated to single subfunction device which is only accessible by applications accessing a given subfunction.

f. Run user space DPDK and kernel driver for a given PF/VF/SF

   A user should be able to run DPDK application for NVF or other user case and kernel driver at the same time for the device.

g. Use existing rich tc offloads interface for networking traffic of PF/VF/SF

   One of the most primary user requirements is to have the ability to reuse the existing tc offloads infrastructure available for PCI PF and VF interfaces. This enables users to perform traffic filter, shaping, QoS configuration at the edge in the NIC itself. This ability enhances the network utilization greatly for east-west traffic between containers or user applications co-located on same server, while still maintaining same level of network security and isolation semantics.

h. Ability to map switch representor with its 'portion of the device'

   A user should have the mapping of switch representor device, its associated devlink port and its host side representation. Such mapping should be available at the device creation time so that necessary switch configuration can be done before a device is attached to container or VM which expects the device to be fully ready when application is operational.

i. Deterministic device naming of rdma and netdevice of PCI PF/VF/SF using systemd/udev [7]
Deterministic device naming is key to have any usable netdevice or rdma device to make use of existing rich network device management software such as NetworkManager [5].

j. Deterministic representor device naming
Currently PCI PF/VF representors naming is well defined using the PCI device name and its phys_port_name device attributes which enables systemd/udev to rename the representor netdevice. A similar naming scheme is necessary so that a resilent system can be built which uses such device naming scheme, for example openvswitch [8] uses it and applies to rules on restarting the openvswitch service.

Now that user requirements are well understood, at software level, it translates to below system software requirements.

1. User should be able to create a 'portion of the device' from its parent device.
2. Instead of creating bulk number of interfaces like SR-IOV num_vfs sysfs knob, user should be able to create one device at a time. This portion of the device is called 'subfunction' for rest of the paper. It is also known as vdev (virtual device) or 'slice' in previous RFCs.
3. User doesn't need to enable SR-IOV to make use of subfunction.
4. Subfunction is 'struct device' where its used in the host guest system with its own PCI BAR resources; such subfunction follows similar probe(), remove() and power management model as described by the linux kernel driver model [6]

A. Devlink overview

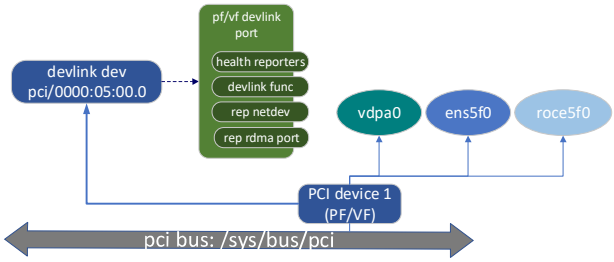


Fig. 1. Devlink eswitch representors and PCI function

- devlink is a PCI level device management interface. It is agnostic of device class such as net/block/rdma/video and more.
- Devlink provides a rich interface for device, port and resource level health reporting and recovery interface to user.
- It offers device level more low-level device parameter configurations which is not directly related to a device class. Though it is mainly used by the network drivers, it is not limited to a class of drivers.
- One of primary advantage of devlink interface is, it is network namespace aware which is syscaller friendly interface; enables more robust way to test kernel and drivers. Being it network namespace aware; it enables user to run management and application in containerized environment which prefers to take benefit of latest software stack and agility of deployment.
- Devlink interface is the primary interface to enable switchdev mode offloads in several networking drivers. This interface is used in the kernel to connect devlink ports, representor netdevice and to provide systemd/udev friendly attributes to deterministically name the representor netdevice.
- Even though devlink has rich interface for eswitch, representor configuration, it is an optional feature. This enables NIC vendors to adopt the interface even without eswitch capability.

B. Current software stack

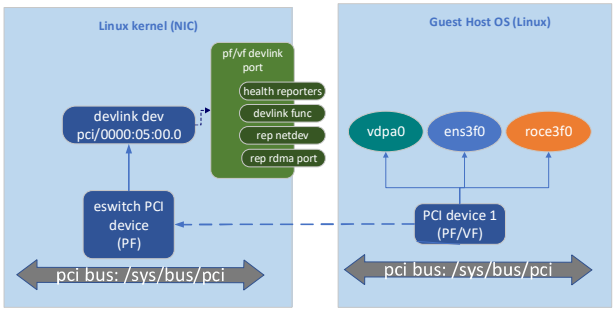


Fig. 2. Devlink eswich representors and PCI device in smartnic based system

In smartnic based system, the eswitch resides on the smartnic running linux kernel. A devlink instance that manages the NIC is residing on the smartnic kernel. In this mode of operation, system administrator or cloud service provider prefers to hot plug in a fully configured subfunction device in the host system.

C. *Why to use devlink interface?*

1. Currently ip link tool is used to set the VF attributes such a {min/max}_tx_rate, mac address. However, it has several limitations.
    a. Its inability to follow create, configure and deploy model
    b. Unable to create, configure and deploy from the smartnic side
    c. Unable to handle non-VF accelerated functions (subfunctions)
    d. Missing linkage with representor switchdev model
2. Why not use a sysfs interface for all the create, configure, deploy user interface?
   sysfs is often used(abused) interface for device or system level parameter configuration. However, it is not suitable for the use cases we described here. Mainly a user interface exposes via non-network namespace aware interface will expose subfunction device parameters to multiple containers which shouldn't have visibility of it.
   It also lacks the ability to report an error, warning string to user for invalid user configuration/attributes; only single error code is reported which doesn't convey user of what attributes in creation/configuration are wrong those should be corrected.
   For scaling to large number of interfaces, creating sysfs files and its inode entries in kernel might be an overkill.
3. Why not use a configfs interface instead of sysfs?
   Configfs also suffers similar limitation as that of sysfs.
   It is also missing all the netlink features that enable to described nested attributes of various data types.
4. Why not to invent a char device ioctl interface?
   It will be reinventing the wheel to achieve all the functionalities available today via devlink and netlink interface of the kernel.

   Devlink interface overcomes all these limitations. It fits with existing usage of SR-IOV switchdev. Devlink provides unified interface to manage PCI physical function, virtual function and subfunctions. Hence, we propose to use devlink interface to subfunction management, whose design is discussed further below in subsequent sections with examples.

III. DESIGN

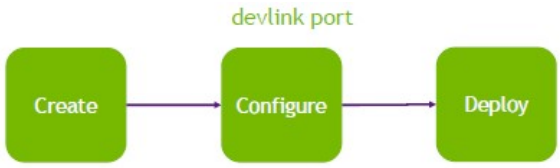


Fig. 3. Subfunction usage flow using devlink port

In this design user follows a simple three step model; here a subfunction is represented using a devlink port. A devlink port represents an eswitch port and a subfunction. This port (and its subfunction) follows create, configure and deploy model.
A creation step involves creating a devlink port. A created port by default is in inactive state. User should do necessary configuration on a the devlink port and devlink port function object.
Once necessary configuration is done, user deploys this port. A deployment step involves creating actual subfunction device on a virtual bus following standard Linux kernel device driver model [6]. This is a hot plug/unplug flow of a device.
In this approach, a device is created, configured and deploy from the eswitch

system through a devlink instance as devlink port. While deployment is done by the NIC (eswitch) system, it can be deployed locally where eswitch resides or on external system where user application is expected to run on the subfunction.

Devlink recently introduced a concept of controller to describe local and external controller that provides a clear annotation of controllers for single host, multi-host and smartnic devices at [3].

I detailed view of the software stack spanning a smartnic kernel and host kernel can be seen below where devlink instance that manages subfunction resides on one system while actual subfunction resides on the external system where this smartnic is plugged in.

It is quite evident that even in non smartnic use cases where delink instance resides on host system; it is equally possible to use the same interface to manage the subfunctions.

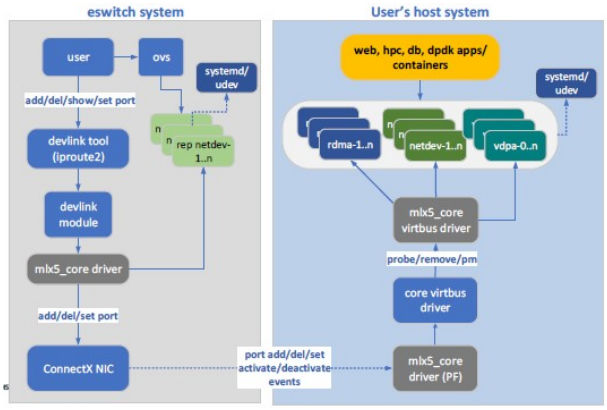
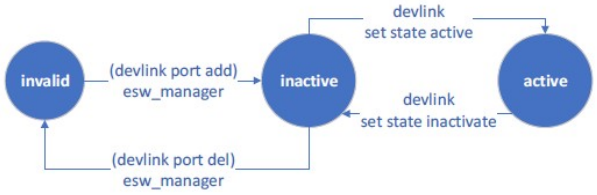


Fig. 4. Detailed software stack on smartnic system for subfunction

Here each subfunction contains a 'struct device' instance on a virtbus. A virtbus is a virtual bus that holds one or more virtual devices with a valid parent device. subfunction utilizes this bus and follows standard driver model [6]. As described further in the example, when subfunction devlink port function is activated, it places the device on the virtbus, triggering standard driver match and probe sequence which creates necessary class devices such as netdevice and/or rdma and/or vdpa device for this subfunction.



IV. DEVLINK EXTENSIONS AND EXAMPLES

Now that the subfunction usage flow and design is clear, this section describes the several commands to manage the subfunction.

A. *Add a subfunction port (on eswitch system)*

Add a subfunction for a PCI PF number 0, where a sufunction's unique number for this devlink instance is 46, whose unique port index is 2.

```
$ devlink port add pci/0000:03:00.0/2 \
        flavour pcisf \
         pfnum 0 sfnum 46
```

B. *Show subfunction port in JSON format*

```
$ devlink port show pci/0000:03:00.0/2 -jp

    “pci/0000:03:00.0/1": {
    "type": "eth“,
    "netdev": " ens5f0pf0sf46",
    "flavour": "pcisf",
    “controller”: 0,
    "pfnum": 0,
    "sfnum": 46,
    “external”: false,
    "function": {
          "hw_addr": "00:11:22:33:44:55",
          "state": "active",
         “opstate": "attached",
    }
}
```

Here external property indicates that this subfunction is for local controller. When a subfunction is plugged into the external controller, its non-zero controller number will be set along with external property being true.

C. *Port function configuration (on eswitch system)*

This configures the subfunction MAC address. When a netdevice of the subfunction is created on the host, it will be this mac address. Once its configured, its state is marked as active. This activation triggers creating the subfunction virtbus device on the host.

```
$ devlink port function set pci/0000:03:00.0/2 \
hw_addr 00:11:22:33:44:55 \
state active
```

Fig. 5. Administrative state of the port function of subfunction devlink port

D. *Subfunction device on the virtbus (in host system)*

When a subfunction is created in host system, such virtbus device will be visible in the sysfs tree of the Linux kernel as:

```
$ ls /sys/bus/virtbus/devices/
mlx5_sf.0
$ cat
/sys/bus/virtbus/devices/mlx5_sf.0/sfnum
46
```

This sysfs attribute helps systemd/udev to renaming the netdevice and rdma device using its parent device and sfnumber. An example netdevice name for this subfunction is enp5s0f0s46.

E. *Devlink show of the subfunction (in host system)*

In below output a subfunction devlink instance mapped to its respective SF can be seen.

```
$ devlink dev show
pci/0000:05:00.0
virtbus/mlx5_sf.0 alias pci/0000:05.00.0%sf46
```

F. *Devlink port show of subfunction (in host system)*

```
$ devlink port show
pci/0000:06:00.0%sf46/0: \
        flavour virtual type eth
        netdev netdev enp3s0f0s46
```

V. STATE MACHINE OF DEVLINK PORT FUNCTION

In this hot plug/unplug usage model, a cloud service provide may want to plug in a subfunction when the host system software may not be fully ready to handle the hot plug event.

In second scenario, an untrusted host software may never honor the hot unplug event and may intent to continue to use the device past its inactivation. Such usage is not desired and such usage shouldn't drive the resource profile on the memory and compute limited smartnic system.

In third scenario, in most graceful usage scenario, a system administrator prefers to inactivate the subfunction devlink port function prior to its deletion.

To address all these scenarios and to provide a clear visibility of it, devlink port function mantains two types of states. One is administrative state, named as just 'state', and other one is 'operational state' that annotates the drivers operational state.

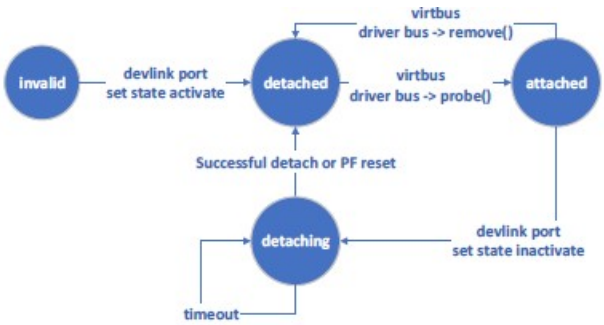


Fig. 6. Operation state diagram of the subfunction driven by subfunction driver and by the eswitch port function state command

VI. OPEN ISSUES

- A system that consist of multiple smartnic hosts and wants to manage multiple smartnic host via Kubernetes orchestration system, doesn't have the visibility at the network level. Since such system consist of multiple controllers in same system, it needs to know its network wide controller id that can be shared with a trusted

network entity which deploys such virtual and subfunctions.

- Virtbus support is still not present in the kernel.
- Ability to match the device provisioning flow for virtual function.
- Selecting which class drivers to be loaded for a given subfunction i.e. defining the device personality at creation time.

VII. CONCLUSION

To handle the lifecycle of the subfunctions for several use cases, devlink offers the most suitable user interface with connection to eswitch offloads. It enables users to deploy subfunctions in unified manner for smartnic and non smartnic use cases. Devlink interface combined with virtbus functionality offers deterministic device naming for RDMA and netdevice interfaces.

VIII. REFERENCES

[1] Devlink subfunction management RFC
https://lore.kernel.org/netdev/20200519092258.GF4655@nanopsycho/

[2] devlink subfunction management plumbing RFC in detail
https://marc.info/?l=linux-netdev&m=158555928517777&w=2

[3] Multi host controller annotation in devlink
https://lore.kernel.org/netdev/20200909045038.63181-1-parav@mellanox.com

[4] Intel's scalable IOV specification
https://software.intel.com/content/www/us/en/develop/download/intel-scalable-io-virtualization-technical-specification.html

[5] NetworkManager
https://github.com/NetworkManager/NetworkManager

[6] Linux kernel driver model
https://www.kernel.org/doc/html/latest/driver-api/driver-model/index.html

[7] systemd/udev
https://github.com/systemd/systemd/tree/master/src/udev

[8] Open vswitch
https://github.com/openvswitch/ovs

[9] rdma device in a net namespace
https://man7.org/linux/man-pages/man8/rdma-dev.8.html