Corigine

Agilio Open vSwitch Offload User Manual

V23.01

Contents:

1 Introduction

1.1 Revision History

Revision	Date	Description
V22.04	29 Apr 2022	Corigine initial public release.
V22.07	30 Jul 2022	Corigine second public release.
V22.10	31 Oct 2022	Corigine third public release.
V23.01	20 Jan 2023	Add Appendix M: UEFI Secure Boot with OutofTree NFP Driver

1.2 About this Guide

This is the User Manual for Agilio Open vSwitch Offload and support provided by Corigine to its customers. The reader can find more elaborated information about the different topics in the links and references provided throughout the document. Bash scripts are indicated with a light blue background.

1.3 Audience

This document is intended for the installer and user of the SmartNIC.

1.4 Contact Us

Corigine Systems, Inc. 2F West, Building 1 No. 1516 Hongfeng Road Wuxing Dist., Huzhou Zhejiang, 313000	
400-615-0098	
https://www.corigine.com/	smartnic-support@corigine.com

2 Abbreviations and Terms

A list of abbreviations and terms used throughout this guide.

Abbreviation/Term	Meaning/Description
BPF	Berkeley Packet Filter
BSP	Board Support Package
COTS	Commercial Off-The-Shelf
DKMS	Dynamic Kernel Module Support
EM	Element Management
HPET	High Precision Event Timer
IOMMU	Input/Output Memory Management Unit
GRE	Generic Routing Encapsulation
KVM	Kernel-based Virtual Machine
MANO	Management and Orchestration
MTU	Maximum Transmission Unit
<netdev>	Network device interface name
<netdev port>	Network device physical port
NFP	Network Flow Processor
NFV	Network Functions Virtualization
NFVI	Network Functions Virtualization Infrastructure
NFVO	Network Functions Virtualization Orchestrator
NIC	Network Interface Card
NM	Network Management
NSD	Network Service Descriptor
NUMA	Non Uniform Memory Access Architecture
OS	Operating System
OOT	Out of Tree

continues on next page

continued from previous page

Abbreviation/Term	Meaning/Description
OVS	Open vSwitch
PCI	Peripheral Component Interconnect
PF	Physical Functions
PNF	Physical Network Functions
PXE	Preboot Execute Environment
RAID	Redundant Arrays of Independent Disks
RSC	Receive Side Coalescing
RSS	Receive Side Scaling
SPOF	Single Points of Failure
SR-IOV	Single Root I/O Virtualization
TSO	TCP Segmentation Offload
UEFI	Unified Extensible Firmware Interface
VDU	Virtualization Deployment Unit
VF	Virtual Functions
VIM	Virtualized Infrastructure Manager
VLAN	Virtual Local Area Network
VNF	Virtualized Network Functions
VNFC	Virtualized Network Functions Component
VNFD	Virtualized Network Functions Descriptor
VNFFG	Virtualized Network Functions Forwarding Graph
VNFM	Virtualized Network Functions Manager
VXLAN	Virtual eXtensible Local Area Network

3 Product Overview

The Agilio family of SmartNICs provide the performance, functionality and programmability required by Cloud operators and service providers struggling to meet performance expectations, without consuming massive CPU cores. The Agilio SmartNICs are available in four options: Agilio CX, Agilio FX, Agilio GX and Agilio LX (https://www.corigine.com/smartnic.html).

3.1 Supported Products

An Agilio SmartNIC product can support different speed types. The following table shows Agilio SmartNIC products that are currently supported and their different supported port speeds.

Supported Agilio product	Supported port speeds
CX 2x25G	2x10G 2x25G 1x10G + 1x25G
CX 2x25G (v2)	2x10G 2x25G 1x10G + 1x25G
GX 2x10G	2x10G
GX 2x25G	2x10G 2x25G 1x10G + 1x25G

3.2 Safety

This section contains “Warnings” and “Cautions”. Warnings are safety related. Failure to follow warnings may lead to injury or equipment damage. Cautions are requirements for proper function. Failure to follow cautions may result in improper operation. All products are low voltage PCIe cards (12V-, 3.3V-supplied per PCIe standard). All lasers in optional transceiver plug-ins are Class 1 or Class 1M. Avoid looking directly at the laser for more than a few seconds.

Warning: Replacements must be performed by qualified personnel only. All installation instructions and requirements specified for the end-use system must be followed.

Caution: None of the units in this document are hot-swappable. Damage will result. Please disconnect all system power feeds before attempting to install or replace any of these products in a system.

Caution: These products may be vulnerable to static electricity (ESD). ESD mitigation controls (e.g. static straps) must be used while handling and installing these products. These products should be stored in antistatic bags or containers when not in use.

3.3 Standards and Regulations

The Agilio SmartNICs adhere to the following regulations.

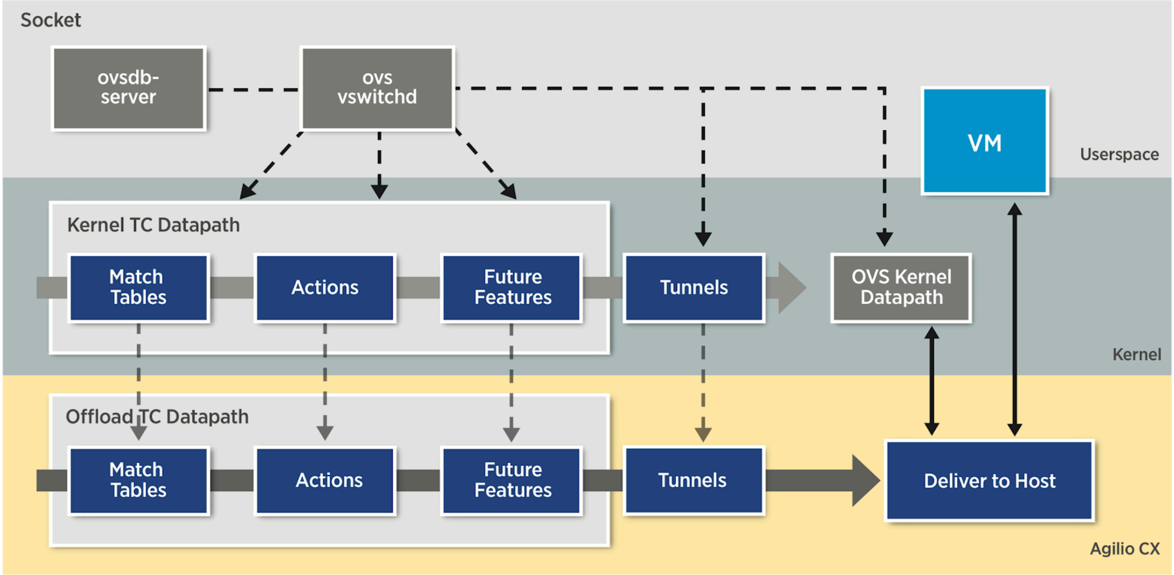
3.3.1 Environmental Compliance

- European Union RoHS II Directive: 2011/65/EU
- European Union REACH Directive: 2006/121/EC
- Administrative Measure on the Control of Pollution Caused by Electronic Information Products (“China ROHS”)
- Congo Conflict Minerals Act of 2009 (Section 1502 of Dodd-Frank Wall Street Reform and Consumer Protection Act including SEC ruling 17 CFR PARTS 240 and 249b)

3.3.2 Regulatory Compliance

- CFR 47 FCC Part 15 Subpart B Class A emissions requirements (USA)
- European Union EMC Directive: 2004/108/EC
- ICES-0003 Issue 4 Class A Digital Apparatus emissions requirements (Canada)
- EN 55022:2010/AC:2011 Class A ITE emissions requirements (EU / CE Mark)
- EN 55024:2010 ITE - immunity characteristics (EU / CE Mark)
- EN 61000-4-2
- EN 61000-4-3
- EN 61000-4-4
- EN 61000-4-6
- EN 61000-4-8
- Kylin Software NeoCertify Certification

4 The Agilio SmartNIC Architecture



The Agilio CX SmartNICs are based on the NFP-4000 and are available in low profile PCIe and OCM v2 NIC form factors suitable for use in COTS servers. This is a 60 core processor with eight cooperatively multithreaded threads per core. The flow processing cores have an instruction set that is optimized for networking. This ensures an unrivaled level of flexibility within the data plane while maintaining performance. The OVS datapath can also be enabled without a server reboot.

Further extensions such as BPF offload, SR-IOV or custom offloads can be added without any hardware modifications or even server reboot. These extensions are not covered by this guide, which deals with the basic and OVS-TC offload firmware only.

The basic firmware offers a wide variety of features including RSS (Receive Side Scaling), Checksum Offload (IPv4/IPv6, TCP, UDP, tx/rx), LSO (Large Segmentation Offload), IEEE 802.3ad, Link flow control, 802.1AZ Link Aggregation, etc. For more details regarding currently supported features, refer to *Basic Firmware Features*.

5 Hardware Installation

This user guide focuses on x86 deployments of Open vSwitch hardware acceleration in supported versions of Ubuntu, Red Hat Enterprise Linux (RHEL), and CentOS. As detailed in *Validating the Driver*, Corigine's Agilio SmartNIC firmware is now upstreamed with the latest supported kernel versions of Ubuntu and RHEL/CentOS. Whilst out-of-tree driver source files are available and installation instructions are included in *Appendix C: Installing the Out-of-Tree NFP Driver*, it is highly recommended, where possible, to make use of the upstreamed drivers. Wherever applicable, separate instructions for RHEL/CentOS and Ubuntu are provided.

Note: All commands in this guide are assumed to be run as root.

5.1 Physical installation

Physically install the SmartNIC in the host server and ensure proper cooling e.g. airflow over card. Ensure the PCI slot is at least Gen3 x8 (can be placed in Gen3 x16 slot). Once installed, power up the server and open a terminal. For additional support, contact smartnic-support@corigine.com.

5.2 Identification

The serial number is printed beside a bar code on the outside of the card and is of the form `SMAAMDAXXXX-XXXXXXXXXXXX`. The `AMDAXXXX` section denotes the assembly ID. In a running system the assembly ID and serial number of a PCI device may be determined using the `ethtool` debug interface. This requires knowledge of the physical function network device identifier, or *<netdev>*, assigned to the SmartNIC under consideration. Consult the section *SmartNIC Netdev Interfaces* for methods on determining this identifier. The interface name *<netdev>* can be otherwise identified using the `ip link` command. The following shell snippet illustrates this method for some particular *<netdev>* whose name is cast as the argument $1:

```
#!/bin/bash
DEVICE=$1
ethtool -W ${DEVICE} 0
DEBUG=$(ethtool -w ${DEVICE} data /dev/stdout | strings)
SERIAL=$(echo "${DEBUG}" | grep "^SN:")
ASSY=$(echo ${SERIAL} | grep -oE AMDA[0-9]{4})
echo ${SERIAL}
echo Assembly: ${ASSY}
```

To run the script execute:

```
# ./<script name> <netdev>
```

Example output of the script:

```
SN: SMAAMDA0099-000117070631 (CX)
Assembly: AMDA0099
```

Note: The `strings` command is commonly provided by the *binutils* package. This can be installed by `yum install binutils` or `apt-get install binutils`, depending on your distribution.

5.3 Validation

Use one of the following `lspci` commands to validate that the SmartNIC is being correctly detected by the host server and identify its PCI address. The PCI vendor identifier for SmartNICs with Board Support Package (BSP) versions before 22.09 is 19ee and the specific PCI vendor identifier for SmartNICs with AMDA2XXX product codes, with a BSP version of at least 22.09 is 1da8. The device tuples are `3800`, `4000` and `6000` respectively, however `3800` devices are currently not supported by the OVS-TC offload firmware.

For SmartNICs with a vendor ID of 19ee:

```
# lspci -bDnnd 19ee:
0000:02:00.0 Ethernet controller [0200]: Netronome Systems, Inc. Device [19ee:4000]
```

Or for SmartNICs with a vendor ID of 1da8:

```
# lspci -bDnnd 1da8:
0000:17:00.0 Ethernet controller [0200]: Corigine, Inc. Device [1da8:3800]
```

Note: The `lspci` command is commonly provided by the *pciutils* package. This can be installed by `yum install pciutils` or `apt-get install pciutils`, depending on your distribution.

6 Validating the Driver

The Corigine SmartNIC physical function driver with support for OVS-TC offload is included in Linux 4.13 and later kernels. The list of minimum required operating system distributions and their respective kernels, which include the NFP driver are as follows:

Operating System	Kernel package version
RHEL/CentOS 7.5	`3.10.0-862.el7`
RHEL/CentOS 7.6	`3.10.0-957.el7`
RHEL/CentOS 7.7	`3.10.0-1062.el7`
RHEL 8.0	`4.18.0-80.el8`
Ubuntu 18.04 LTS	`4.15.0-20.21`

Note: Only the x86_64 architecture has been verified. If support for other architectures are required, please contact Corigine support: *Contact Us*.

6.1 Confirm Upstreamed NFP Driver

Use the `modinfo` command to confirm that your current operating system includes the upstreamed `nfp` module:

```
# modinfo nfp | head -3
filename:
/lib/modules/3.10.0-862.el7/kernel/drivers/net/ethernet/netronome/nfp/nfp.ko.xz
description:    The Netronome Flow Processor (NFP) driver.
license:        GPL
```

Note: If the module is not found in your current kernel, refer to *Appendix C: Installing the Out-of-Tree NFP Driver* for more instructions, or upgrade your distribution and kernel to a version that includes the upstreamed drivers.

6.2 Confirm that the NFP Driver is Loaded

Use the `lsmod` command to list the loaded driver modules and `grep` to search for the `nfp` string:

```
# lsmod | grep nfp
nfp                   161364  0
```

If the NFP driver is not loaded, the following command loads it manually:

```
# modprobe nfp
```

Note: If the driver cannot be loaded after the OOT driver is installed in the case of UEFI secure boot enable, please refer to *Appendix M: UEFI Secure Boot with Out-of-Tree NFP Driver*.

7 Validating the Firmware

Corigine SmartNICs are fully programmable devices and depend on the driver to load firmware onto the device at runtime. It is important to note that the functionality of the SmartNIC significantly depends on the firmware loaded. The firmware files should be present in the following directory (contents may vary depending on the installed firmware and distribution layout):

```
# ls -ogR --time-style="+" /lib/firmware/netronome/
/lib/firmware/netronome/:
total 8
drwxr-xr-x. 2 4096  flower
drwxr-xr-x. 2 4096  nic
lrwxrwxrwx  1   31  nic_AMDA0081-0001_1x40.nffw -> nic/nic_AMDA0081-0001_1x40.nffw
lrwxrwxrwx  1   31  nic_AMDA0081-0001_4x10.nffw -> nic/nic_AMDA0081-0001_4x10.nffw
lrwxrwxrwx  1   31  nic_AMDA0096-0001_2x10.nffw -> nic/nic_AMDA0096-0001_2x10.nffw
lrwxrwxrwx  1   31  nic_AMDA0097-0001_2x40.nffw -> nic/nic_AMDA0097-0001_2x40.nffw
lrwxrwxrwx  1   36  nic_AMDA0097-0001_4x10_1x40.nffw -> nic/nic_AMDA0097-0001_4x10_
↪1x40.nffw
lrwxrwxrwx  1   31  nic_AMDA0097-0001_8x10.nffw -> nic/nic_AMDA0097-0001_8x10.nffw
lrwxrwxrwx  1   36  nic_AMDA0099-0001_1x10_1x25.nffw -> nic/nic_AMDA0099-0001_1x10_
↪1x25.nffw
lrwxrwxrwx  1   31  nic_AMDA0099-0001_2x10.nffw -> nic/nic_AMDA0099-0001_2x10.nffw
lrwxrwxrwx  1   31  nic_AMDA0099-0001_2x25.nffw -> nic/nic_AMDA0099-0001_2x25.nffw
lrwxrwxrwx  1   34  pci-0000:04:00.0.nffw -> flower/nic_AMDA0097-0001_2x40.nffw
lrwxrwxrwx  1   34  pci-0000:06:00.0.nffw -> flower/nic_AMDA0096-0001_2x10.nffw

/lib/firmware/netronome/flower:
total 11692
lrwxrwxrwx. 1      17  nic_AMDA0081-0001_1x40.nffw -> nic_AMDA0097.nffw
lrwxrwxrwx. 1      17  nic_AMDA0081-0001_4x10.nffw -> nic_AMDA0097.nffw
lrwxrwxrwx. 1      17  nic_AMDA0096-0001_2x10.nffw -> nic_AMDA0096.nffw
-rw-r--r--. 1 3987240  nic_AMDA0096.nffw
lrwxrwxrwx. 1      17  nic_AMDA0097-0001_2x40.nffw -> nic_AMDA0097.nffw
lrwxrwxrwx. 1      17  nic_AMDA0097-0001_4x10_1x40.nffw -> nic_AMDA0097.nffw
lrwxrwxrwx. 1      17  nic_AMDA0097-0001_8x10.nffw -> nic_AMDA0097.nffw
-rw-r--r--. 1 3988184  nic_AMDA0097.nffw
lrwxrwxrwx. 1      17  nic_AMDA0099-0001_2x10.nffw -> nic_AMDA0099.nffw
lrwxrwxrwx. 1      17  nic_AMDA0099-0001_2x25.nffw -> nic_AMDA0099.nffw
-rw-r--r--. 1 3990552  nic_AMDA0099.nffw

/lib/firmware/netronome/nic:
total 12220
-rw-r--r--. 1 1380496  nic_AMDA0081-0001_1x40.nffw
-rw-r--r--. 1 1389760  nic_AMDA0081-0001_4x10.nffw
-rw-r--r--. 1 1385608  nic_AMDA0096-0001_2x10.nffw
```

(continues on next page)

(continued from previous page)

```
-rw-r--r--. 1 1385664  nic_AMDA0097-0001_2x40.nffw
-rw-r--r--. 1 1391944  nic_AMDA0097-0001_4x10_1x40.nffw
-rw-r--r--. 1 1397880  nic_AMDA0097-0001_8x10.nffw
-rw-r--r--. 1 1386616  nic_AMDA0099-0001_1x10_1x25.nffw
-rw-r--r--. 1 1385608  nic_AMDA0099-0001_2x10.nffw
-rw-r--r--. 1 1386368  nic_AMDA0099-0001_2x25.nffw
```

If `netronome/flower` is not present, the `linux-firmware` package on the system is probably outdated and does not contain the upstreamed OVS-TC firmware. Refer to *Appendix G: Upgrading TC Firmware* for upgrade instructions. The NFP driver will search for firmware in `/lib/firmware/netronome` in the following order:

```
1: serial-_SERIAL_.nffw
2: pci-_PCI_ADDRESS_.nffw
3: nic-_ASSEMBLY-TYPE___BREAKOUTxMODE_.nffw
```

This search is logged by the kernel when the driver is loaded. For example:

```
# dmesg | grep -A 4 nfp.*firmware
[   3.260788] nfp 0000:04:00.0: nfp: Looking for firmware file in order of priority:
[   3.260810] nfp 0000:04:00.0: nfp:   netronome/serial-00-15-4d-13-51-0c-10-ff.
↪nffw: not found
[   3.260820] nfp 0000:04:00.0: nfp:   netronome/pci-0000:04:00.0.nffw: not found
[   3.262138] nfp 0000:04:00.0: nfp:   netronome/nic_AMDA0097-0001_2x40.nffw: found,
↪ loading...
```

The version of the loaded firmware for a particular netdev interface, as found in *SmartNIC Netdev Interfaces* (for example `enp4s0`), or a physical port representor (for example, `enp4s0np0`) can be displayed with the `ethtool` command:

```
# ethtool -i <netdev>
driver: nfp
version: 3.10.0-862.el7.x86_64 SMP mod_u
firmware-version: 0.0.3.5 0.20 nic-2.0.7 nic
expansion-rom-version:
bus-info: 0000:04:00.0
```

Note: Replace <netdev> with the machine's specific interface associated with the SmartNIC's PF, which is expected to be something like `enp4s0` or `enp4s0np0`.

Firmware versions are displayed in order: NFD version, NSP version, APP FW version, driver APP. The specific output above shows that basic NIC firmware is running on the card, as indicated by `nic` in the `firmware-version` field.

8 Selecting the TC Offload Firmware

In order to initialize the SmartNIC with the TC offload firmware, a symbolic link based on the PCI address of the SmartNIC should be created to the desired firmware. When the kernel module is loaded, it will load the specified firmware instead of the default CoreNIC firmware. The TC offload firmware is located in the `/lib/firmware/netronome/flower/` directory.

Create a bash script named `agilio-tc-fw-select.sh` which will be used to create and persist this symbolic link.

The script would need the <*netdev*> number. Review *SmartNIC Netdev Interfaces* to identify the SmartNIC's <*netdev*>, for example ens4np0, which is the machine's specific interface associated with the SmartNIC's physical port. The interface <*netdev*> can be otherwise identified using the `ip link` command:

```
# ip link
11: ens3np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP␣
↪mode DEFAULT group default qlen 1000
    link/ether 00:15:4d:13:00:8e brd ff:ff:ff:ff:ff:ff
    altname enp2s0np0
12: ens3np1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP␣
↪mode DEFAULT group default qlen 1000
    link/ether 00:15:4d:13:00:8f brd ff:ff:ff:ff:ff:ff
    altname enp2s0np1
13: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP mode␣
↪DEFAULT group default qlen 1000
    link/ether f6:f8:98:ab:98:31 brd ff:ff:ff:ff:ff:ff
14: ens5np0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode␣
↪DEFAULT group default qlen 1000
    link/ether 00:15:4d:13:06:0d brd ff:ff:ff:ff:ff:ff
    altname enp5s0np0
```

Insert the following in the `agilio-tc-fw-select.sh` script:

```
#!/bin/bash
DEVICE=${1}
DEFAULT_ASSY=scan
ASSY=${2:-${DEFAULT_ASSY}}
APP=${3:-flower}

if [ "x${DEVICE}" = "x" -o ! -e /sys/class/net/${DEVICE} ]; then
    echo Syntax: ${0} device [ASSY] [APP]
    echo
    echo This script associates the TC Offload firmware
    echo with a Corigine SmartNIC.
    echo
    echo device: is the network device associated with the SmartNIC
    echo ASSY: defaults to ${DEFAULT_ASSY}
    echo APP: defaults to flower. flower-next is supported if updated
    echo      firmware has been installed.
    exit 1
fi

# It is recommended that the assembly be determined by inspection
# The following code determines the value via the debug interface
if [ "${ASSY}x" = "scanx" ]; then
    ethtool -W ${DEVICE} 0
    DEBUG=$(ethtool -w ${DEVICE} data /dev/stdout | strings)
    SERIAL=$(echo "${DEBUG}" | grep "^SN:")
    ASSY=$(echo ${SERIAL} | grep -oE AMDA[0-9]{4})
fi

PCIADDR=$(basename $(readlink -e /sys/class/net/${DEVICE}/device))
FWDIR="/lib/firmware/netronome"

# AMDA0081 and AMDA0097 uses the same firmware
if [ "${ASSY}" = "AMDA0081" ]; then
    if [ ! -e ${FWDIR}/${APP}/nic_AMDA0081.nffw ]; then
       ln -sf nic_AMDA0097.nffw ${FWDIR}/${APP}/nic_AMDA0081.nffw
   fi
fi

FW="${FWDIR}/pci-${PCIADDR}.nffw"
ln -sf "${APP}/nic_${ASSY}.nffw" "${FW}"

# insert distro-specific initramfs section here...
```

For RHEL 7.5+ and CentOS 7.5+ systems, it is recommended to append the following snippet:

```
# RHEL 7.5+ and CentOS 7.5+ distro-specific initramfs section
DRACUT_CONF=/etc/dracut.conf.d/98-nfp-firmware.conf
echo "install_items+=\" ${FW} \"" > "${DRACUT_CONF}"
dracut -f
```

Alternatively, for Ubuntu 18.04 systems, append the following snippet, instead:

```
# Ubuntu 18.04 distro-specific initramfs section
HOOK=/etc/initramfs-tools/hooks/agilio_firmware
cat >${HOOK} << EOF
#!/bin/sh
PREREQ=""
prereqs()
{
    echo "\$PREREQ"
}
case "\$1" in
prereqs)
    prereqs
    exit 0
    ;;
esac
. /usr/share/initramfs-tools/hook-functions
cp "${FW}" "\${DESTDIR}${FW}"
if have_module nfp ; then
    manual_add_modules nfp
fi
exit 0
EOF
chmod a+x "${HOOK}"
update-initramfs -u
```

This adds the `symlink` and firmware to the `initramfs`.

The script needs execute permission which is given with the following command:

```
# chmod +x agilio-tc-fw-select.sh
```

If the user wishes to auto-detect the Assembly ID, run the script, `./agilio-tc-fw-select.sh`, and reload the driver with the following commands:

```
# ./agilio-tc-fw-select.sh <netdev> scan
# rmmod nfp; modprobe nfp
```

If the out-of-tree firmware repository has been installed (as described in *Appendix G: Upgrading TC Firmware*) and the user wishes to select that instead, run the script and reload the driver with the commands:

```
# ./agilio-tc-fw-select.sh <netdev> scan flower-next
# rmmod nfp; modprobe nfp
```

8.1 Verify Firmware is Loaded

The firmware should indicate that it has the FLOWER capability. This can be confirmed by inspecting the kernel message buffer using `dmesg`:

```
# dmesg | grep nfp
[ 3131.714215] nfp 0000:04:00.0 eth4: Netronome NFP-6xxx Netdev: TxQs=8/8 RxQs=8/8
[ 3131.714221] nfp 0000:04:00.0 eth4: VER: 0.0.5.5, Maximum supported MTU: 9420
[ 3131.714227] nfp 0000:04:00.0 eth4: CAP: 0x20140673 PROMISC RXCSUM TXCSUM RXVLAN GATHER TSO1 RSS2 AUTOMASK IRQMOD FLOWER
```

Loading of flower firmware may also be confirmed using `ethtool`. `AOTC` indicates that OVS-TC firmware was loaded, as does `flow`. e.g.:

```
# ethtool -i <netdev>
driver: nfp
version: 3.10.0-862.el7.x86_64 SMP mod_u
firmware-version: 0.0.5.5 0.22 0AOTC28A.5642 flow
expansion-rom-version:
bus-info: 0000:04:00.0
```

Note: Replace <*netdev*> with the machine's specific interface associated with the SmartNIC's PF, which is expected to be something like `ens3np0`.

9 SmartNIC Netdev Interfaces

9.1 Representors

Representor <*netdevs*>, or representors, are <*netdevs*> created to represent the switch-side of a port. When Flower firmware for Agilio CX SmartNIC is loaded the following <*netdevs*> are created:

- A <*netdev*> for the PCI physical function (PF) to represent the PCI connection between the host and the card.
- Representor <*netdevs*> for each physical port (MAC) of the card. These are used, for example, to configure the link state of the port, access port statistics and carry fallback traffic. Fallback traffic are packets which are not handled by the datapath on the SmartNIC, usually because there is no matching rule present in the flow-cache, and thus they are sent to the host for processing.
- A representor <*netdev*> for the PF. This is not currently used in an OVS-TC system.

When SR-IOV VFs (virtual functions) are instantiated, a representor <*netdev*> is created for each VF. Like representors for physical ports, these are used for configuration, statistics and fallback packets. When using OVS-TC it is the physical port representor <*netdevs*>, and VF representor <*netdevs*> that are attached to OVS which then allow OVS to configure the associated ports and VFs to send and receive fallback packets.

9.2 Identification

To identify the Agilio NIC interfaces, begin by identifying the physical function and physical port representor names. This may be determined by examining the <*netdevs*> of the PF PCI devices for the Agilio NIC using the `lspci` tool. The `lspci` tool requires the device tuple with the specific Corigine vendor. The vendor ID is either `19ee` or `1da8` and the device tuples are `4000`, `3800` or `6000` respectively. These variables are indicated to the `lspci` tool in the form <vendor>:<device tuples> tuples (e.g. `19ee:4000` or `1da8:3800`). The <*netdevs*> associated with these devices may be determined by examining sysfs.

For SmartNICs with a vendor ID of 19ee:

```
#!/bin/bash
BDFS=$({ lspci -bDnnd 19ee:; } | cut -f 1 -d " ")
for i in $BDFS; do ls /sys/bus/pci/drivers/nfp/$i/net/; done
```

Or for SmartNICs with a vendor ID of 1da8:

```
#!/bin/bash
BDFS=$({ lspci -bDnnd 1da8:; } | cut -f 1 -d " ")
for i in $BDFS; do ls /sys/bus/pci/drivers/nfp/$i/net/; done
```

An example output of this would be:

```
enp4s0np0 enp4s0np1 enp4s0
```

Where `enp4s0np0` and `enp4s0np1` are the physical port representors and `enp4s0` is the physical function <*netdev*>:

The naming scheme for each port and physical function is dependent on the motherboard and the PCI slot into which the NFP is installed. The PF name should be that associated with the PCI slot and the physical port representor names should be the PF name with `np[x]` appended.

Note: Platform and BIOS configuration as well as enabling `biosdevname` can affect the port naming policies.

To confirm that the representor `enp4s0np0` is a physical port, verify the contents of the following file in sysfs:

```
# cat /sys/class/net/enp4s0np0/phys_port_name
p0
```

The physical ports will report the physical port name, while the physical function (in this case `enp4s0`) will report an error.

```
# cat /sys/class/net/enp4s0/phys_port_name
cat /sys/class/net/enp4s0/phys_port_name: Operation not supported
```

Once a physical port name has been determined, it is possible to determine the `phys_switch_id` of the NFP. This is required to determine the names of the VF representors when multiple NFPs are installed in a host. If an NFP has more than one physical port, both ports will share the same `phys_switch_id`. The PF will report an error when its `phys_switch_id` is queried. For example, the `phys_switch_id` of the device for which `enp4s0np0` is a physical port , is:

```
# cat /sys/class/net/enp4s0np0/phys_switch_id
00154d13510c
```

Please refer to the section *Configuring SR-IOV* for information on how to instantiate VFs.

To identify VF representors, query the devices listed in `/sys/class/net` for `phys_port_name` and `phys_switch_id`. VFs will share the switch id and report their individual VF number in the form `p0vf[x]`. The following script creates a translation variable in bash that translates from VF index to interface name:

```
#!/bin/bash
declare -A vf_repr_ifname
for ifname in $(ls /sys/class/net); do
    pn=$(cat /sys/class/net/${ifname}/phys_port_name 2> /dev/null)
    [ "x${pn}" != "x" ] || continue
    vfidx=$(echo "${pn}" | sed -rn 's/pf0vf([0-9]+)$/\1/p')
    [ "x${vfidx}" != "x" ] || continue
    vf_repr_ifname[${vfidx}]="${ifname}"
done
```

Note: This operation is not atomic and so any other subsystem that renames the network devices may invalidate this table.

The virtual functions associated with a PF PCI address are symlinked into the sysfs directory associated with the PF PCI device. For example, if the PF is located at `0000:04:00.0`, `VF1` would be at `0000:04:08.1`, and `VF9` would be at `000:04:09.1`. In `/sys/bus/pci/devices/0000:04:00.0/ virtfn0` and `virtfn9` would link to those addresses:

```
# ls -og --time-style="+" /sys/bus/pci/drivers/nfp/0000:04:00.0/virtfn[19]
lrwxrwxrwx 1 0  /sys/bus/pci/drivers/nfp/0000:04:00.0/virtfn1 -> ../0000:04:08.1
lrwxrwxrwx 1 0  /sys/bus/pci/drivers/nfp/0000:04:00.0/virtfn9 -> ../0000:04:09.1
```

9.3 Support for `biosdevname`

Corigine NICs support `biosdevname` *<netdev>* naming with recent versions of the utility, circa December 2018, e.g. RHEL 8.0 and up. Furthermore, `biosdevname` will only be supported on kernel v4.19+. There are some notable points to be aware of:

- Whenever an unsupported *<netdev>* is considered for naming, the `biosdevname` naming will be skipped and the next inline naming scheme will take preference, e.g. the `systemd` naming policies.
- *<Netdevs>* in breakout mode are not supported for naming.
- VF *<netdevs>* will still be subject to `biosdevname` naming irrespective of the breakout mode of other *<netdevs>*.
- Physical function *<netdevs>* are not supported for naming.
- PF and VF representor *<netdevs>* are not supported for naming.
- When using an older version of the `biosdevname` utility or an older kernel, users will observe inconsistent naming of *<netdevs>*.

To disable `biosdevname` users can add `biosdevname=0` to the kernel command line.

Refer to the online `biosdevname` documentation for more details about the naming policy convention that will be applied.

10 PF Link Configuration

The physical function *<netdev>* for the PCI device acts as a lower-device for representors and must be up in order to allow sending and receiving fallback traffic on representors. As the PF *<netdev>* is not used directly to carry packets, it is recommended that it be brought up without an IP address. It is also advised to set the maximum transmission unit for the PF interface to the largest value supported by the firmware, as advertised in the kernel message buffer, to avoid fallback packets from being unnecessarily dropped due to being larger than the MTU of the PF.

```
# dmesg | grep MTU
[ 3131.714221] nfp 0000:04:00.0 eth4: VER: 0.0.5.5, Maximum supported MTU: 9420
```

10.1 Settings

10.1.1 RHEL 7.5+ and CentOS 7.5+

`iproute` may be configured to bring up a device without addresses as follows. `iproute` may not present on some installs, it can be installed using `yum`:

```
# yum install iproute
```

In this example, the device is `enp4s0` (replace this to match the PF *<netdev>* in question).

```
# ip address flush dev enp4s0 scope global
# ip link set dev enp4s0 mtu 9240
```

This process creates a connection for the *<netdev>*, disables the IPv4 configuration, sets the IPv6 configuration to be ignored and finally sets the MTU of the PF to the maximum value supported by the firmware in order to avoid drops of fallback packets.

`iproute` may now be used to bring up the connection. This will bring up the link on the physical function which is essential to allow communication between the TC offload mechanism and the NFP.

```
# ip link set dev enp4s0 up
```

Note: It is recommended to prevent `NetworkManager` from managing all NFP interfaces other than the PF. Having `NetworkManager` manage the representor interfaces can interfere with the operation of OVS-TC. An example of how to correctly configure `NetworkManager` can be found at *Confirming Connectivity*

10.1.2 Ubuntu

A `networkd-dispatcher` script can be used to set an interface's MTU and bring up the link of the PF's *<netdev>* without adding any IP addresses to it. Reconfiguring the MTU is discussed in more detail in *Configuring Interface Maximum Transmission Unit (MTU)*. In this example, a simple script is run for each routable interface. Again, the device used here is `enp4s0` which should be changed to match the PF *<netdev>* installed in the system.

```
#!/bin/sh
cat > /usr/lib/networkd-dispatcher/routable.d/50-ifup-noaddr << 'EOF'
#!/bin/sh
ip link set mtu 9420 dev enp4s0
ip link set up dev enp4s0
EOF
chmod u+x /usr/lib/networkd-dispatcher/routable.d/50-ifup-noaddr
```

In order to ensure the hook above is run, regardless of whether `networkd-dispatcher` runs before or after `systemd-networkd`, the configuration of `networkd-dispatcher` should be updated to generate events reflecting the existing state and behavior when it starts up. This is the `--run-startup-triggers` option and may be passed to `networkd-dispatcher` on start-up by adding it to `/etc/default/networkd-dispatcher`.

```
#!/bin/sh
cat > /etc/default/networkd-dispatcher << 'EOF'
# Specify command line options here. This config file is used
# by the included systemd service file.
networkd_dispatcher_args="--run-startup-triggers"
EOF
```

Restarting `network-dispatcher` should now set the MTU and bring up the link of `p1p5` if there are any routable interfaces.

Note: For Ubuntu based systems, VF creation may also be done using this trigger method. Refer to *Configuring SR-IOV* for details.

```
# systemctl restart networkd-dispatcher
```

The service status of `networkd-dispatcher` will then reflect the changes implemented:

```
# service networkd-dispatcher status
  networkd-dispatcher.service - Dispatcher daemon for systemd-networkd
      Loaded: loaded (/lib/systemd/system/networkd-dispatcher.service; enabled;␣
↪vendor preset: enabled)
      Active: active (running) since Wed 2018-05-16 13:05:48 UTC; 2min 31s ago
      Main PID: 41757 (networkd-dispat)
```

(continues on next page)

(continued from previous page)

```
        Tasks: 2 (limit: 7372)
              CGroup: /system.slice/networkd-dispatcher.service
                        └─41757 /usr/bin/python3 /usr/bin/networkd-dispatcher --
↪run-startup-triggers
```

10.1.3 Upping Physical Port Representors

When using `libvirt` to manage virtual machines on the host, it's also highly recommended to up all physical port representors, whether or not they are plugged into the physical network. This is because `libvirt` expects to manage the virtual functions using any *<netdev>* associated with them. The specific *<netdev>* chosen depends on which is listed first in `sysfs`. Since it's very hard to control this, the recommended procedure is to apply the above procedure to all the *<netdevs>* associated with the PF.

10.2 Verification

Verify link state and MTU of the PF *<netdev>*. For example the *<netdev>* `enp4s0` (unlike the physical port representors `enp4s0np0` or `enp4s0np1`) outputs:

```
# ip addr show enp4s0
14: enp4s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9420 qdisc mq state UP group␣
↪default qlen 1000
    link/ether 0e:c4:88:90:27:88 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::cc4:88ff:fe90:2788/64 scope link
      valid_lft forever preferred_lft forever
```

11 Install Open vSwitch

11.1 Installation From a Recent Distribution

The preferred method of installing and upgrading Open vSwitch is through the distribution repositories. The minimum recommended versions are those provided in supported releases of distributions. As a guide they are as follows:

Operating System	OVS Version
CentOS 7.6	2.9.0
CentOS 8.0	2.11.0
Ubuntu 18.04 LTS	2.10.0

11.1.1 RHEL

Please refer to *Appendix B: Red Hat Repositories* for information on configuring Red Hat repositories. Once the repositories are configured, install Open vSwitch using `yum`:

```
# yum install -y openvswitch
```

11.1.2 CentOS

For CentOS it is recommended to add OpenStack repositories from RDO. This can be achieved by using `yum` directly. First install `yum-utils` to get the `yum-config-manger` utility, then install the repository:

```
# yum install -y yum-utils
# yum install -y centos-release-openstack-wallaby
```

It is recommended to disable this repository by default and only enable it for the Open vSwitch install:

```
# yum-config-manager --disable centos-openstack*
```

Install Open vSwitch by temporarily enabling the repository for the specific `yum` call:

```
# yum install -y --enablerepo centos-openstack-wallaby openvswitch
```

At the time of writing this will install `openvswitch-2.15.4`.

11.1.3 Ubuntu

In Ubuntu, Open vSwitch can be installed using `apt-get`:

```
# apt-get update
# apt-get install -y openvswitch-switch
```

11.1.4 Check OVS Install

If the installation procedure completed successfully, `systemctl status openvswitch.service` should return the service status. More information on using Open vSwitch is provided later in *Using Open vSwitch*.

12 Using the Linux Driver

12.1 Configuring SR-IOV

To configure Single Root I/O Virtualization (SR-IOV) virtual functions, ensure that SR-IOV is enabled in the BIOS of the host machine. If SR-IOV is disabled or unsupported by the motherboard/chipset being used, the kernel message log will contain a `PCI SR-IOV:-12` error when trying to create a VF. This can be queried using the `dmesg` tool. The number of supported virtual functions on a `netdev` is exposed by `sriov_totalvfs` in `sysfs`. For example, if `ens3` is the interface associated with the SmartNIC's physical function, the following command will return the total supported number of VF's:

```
# cat /sys/class/net/ens3/device/sriov_totalvfs
55
```

Virtual functions can be allocated to a network interface by writing an integer to the `sysfs` file. For example, to allocate 16 virtual functions to `ens3`:

```
# echo 16 > /sys/class/net/ens3/device/sriov_numvfs
```

Note: The current Corigine cards supporting TC offload only have a single PF. This means that, even though the SR-IOV interfaces are exposed on the PF `netdev` and the physical port representors, they refer to the same underlying physical function. It is therefore an error to attempt to allocate VF's to multiple physical port representors.

See *Open vSwitch Hardware Offload Example* for a practical application. SR-IOV Virtual functions cannot be re-allocated dynamically. In order to change the number of allocated virtual functions, existing functions must first be deallocated by writing a 0 to the `sysfs` file. Otherwise, the system will return a device or resource busy error:

```
# echo 0 > /sys/class/net/ens3/device/sriov_numvfs
```

Note: Ensure any VMs are shut down and applications that may be using the VFs are stopped before deallocation.

To persist the virtual functions on the system, it is suggested that the system initialization scripts be updated to manage them. The following snippet illustrates how to implement such a configuration with the physical function `ens3`:

```
#!/bin/sh
cat > /etc/init.d/vf-init << 'EOF'
#!/bin/sh
ip link set mtu 9420 dev ens3
ip link set up dev ens3
echo 4 > /sys/class/net/ens3/device/sriov_numvfs
EOF
chmod u+x /etc/init.d/vf-init
```

Executing the script above will ensure that, when the system is booted, 4 VF interfaces connected to the PF on ens3 will be created.

In Ubuntu systems, `networkd-dispatcher` can be used, as demonstrated below:

```
#!/bin/sh
cat > /usr/lib/networkd-dispatcher/routable.d/50-ifup-noaddr << 'EOF'
#!/bin/sh
ip link set mtu 9420 dev ens3
ip link set up dev ens3
cat /sys/class/net/ens3/device/sriov_totalvfs > \
/sys/class/net/ens3/device/sriov_numvfs
EOF
chmod u+x /usr/lib/networkd-dispatcher/routable.d/50-ifup-noaddr
```

12.2 Configuring Interface Media Mode

This section details the configuration of the SmartNIC physical interfaces.

Note: For older kernels that do not support the configuration methods outlined below, please refer to *Appendix D: Working with Board Support Package* on how to make use of the BSP toolkit to configure interfaces.

12.2.1 Configuring Interface Link-speed

The following steps explain how to change between 10G mode and 25G mode on CX 2x25GbE cards. The changing of port speed must be done in order, port 0 (p0) must be set to 10G before port 1 (p1) may be set to 10G.

Down respective interface(s):

```
# ip link set dev <netdev> down
```

Set interface link speed to 10G:

```
# ethtool -s <netdev> speed 10000
```

Alternatively, set interface link speed to 25G:

```
# ethtool -s <netdev> speed 25000
```

Reload driver for changes to take effect:

```
# rmmod nfp && modprobe nfp
```

Note: Replace `<netdev>` with the machine's specific interface associated with the SmartNIC's PF, which is expected to be something like `enp4s0` or `enp4s0np0`.

12.3 Configuring Interface Maximum Transmission Unit (MTU)

The MTU of interfaces can temporarily be set using the `iproute2` or `ifconfig` tools. Note that this change will not persist. Setting this via Network Manager, or other appropriate OS configuration tool, is recommended. Hereafter is an example of setting the MTU with the `ip link` command, which is part of the `iproute2` package.

Set MTU of `<netdev>` interface to 9000 bytes:

```
# ip link set dev <netdev> mtu 9000
```

It is the responsibility of the user or the orchestration layer to set appropriate MTU values when handling jumbo frames or utilizing tunnels. For example, if packets sent from a VM are to be encapsulated on the card and egress a physical port, then the MTU of the VF should be set to lower than that of the physical port to account for the extra bytes added by the additional header.

If a setup is expected to see fallback traffic between the SmartNIC and the kernel, then the user should also ensure that the PF MTU is appropriately set to avoid unexpected drops on this path.

12.4 Configuring FEC modes

CX 2x25GbE SmartNICs support FEC mode configuration, e.g. *Auto*, *Firecode Base-R*, *Reed-Solomon* and *Off* modes. Each physical port's FEC mode can be set independently via the `ethtool` command. To view the currently supported FEC modes of the interface use the following:

```
# ethtool <netdev>
Settings for <netdev>:
    Supported ports: [ FIBRE ]
    Supported link modes:   Not reported
    Supported pause frame use: No
    Supports auto-negotiation: No
    Supported FEC modes: None BaseR RS
```

(continues on next page)

(continued from previous page)

```
    Advertised link modes:  Not reported
    Advertised pause frame use: No
    Advertised auto-negotiation: No
    Advertised FEC modes: BaseR RS
    Speed: 25000Mb/s
    Duplex: Full
    Port: Direct Attach Copper
    PHYAD: 0
    Transceiver: internal
    Auto-negotiation: on
    Link detected: yes
```

The output above details which FEC modes are supported for this interface. Note that the CX 2x25GbE SmartNIC used for the example above only supports *Firecode Base-R* FEC mode on ports that are forced to 10G speed.

Note: `ethtool` FEC support is only available in kernel 4.14 and newer or RHEL 7.5+ CentOS 7.5, and equivalent distributions. The Corigine upstream kernel driver provides `ethtool` FEC support from kernel 4.15. Furthermore, the SmartNIC NVRAM versioning system has been updated, and it is recommended for the NVRAM version to be at least 22.04-0. With respect to the previous naming, a minimum NVRAM version of 020025.020025.02006e is required to support `ethtool` FEC get/set operations.

To determine your version of the current SmartNIC NVRAM, examine the kernel message buffer:

```
# dmesg | grep 'nfp.*BSP'
[2.944857] nfp 0000:65:00.0: BSP: 22.10-0
```

This example lists a version of 22.10-0 which is sufficient to support `ethtool` FEC mode configuration. To update your SmartNIC NVRAM flash, please contact Corigine support.

If the SmartNIC NVRAM or the kernel does not support `ethtool` modification of FEC modes, no supported FEC modes will be listed in the `ethtool` output for the port. This could be because of an outdated kernel version or an unsupported distribution (e.g. Ubuntu 16.04, irrespective of the kernel version).

```
# ethtool <netdev>
Settings for <netdev>:
...
Supported FEC modes: None
```

To show the currently active FEC mode for either the netdev or the physical port representors:

```
# ethtool --show-fec <netdev>
FEC parameters for <netdev>:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: Auto
```

To force the FEC mode for a particular port, autonegotiation must be disabled with the following:

```
# ip link set <netdev> down
# ethtool -s <netdev> autoneg off
# ip link set <netdev> up
```

Note: In order to change the autonegotiation configuration the port must be down.

Note: Changing the autonegotiation configuration will not affect the SmartNIC port speed. Please see *Configuring Interface Link-speed* to adjust this setting.

To modify the FEC mode to *Firecode Base-R*:

```
# ethtool --set-fec <netdev> encoding baser
```

Verify the newly selected mode:

```
# ethtool --show-fec <netdev>
FEC parameters for <netdev>:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: BaseR
```

To modify the FEC mode to *Reed-Solomon*:

```
# ethtool --set-fec <netdev> encoding rs
```

Verify the newly selected mode:

```
# ethtool --show-fec <netdev>
FEC parameters for <netdev>:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: RS
```

To modify the FEC mode to *Off*:

```
# ethtool --set-fec <netdev> encoding off
```

Verify the newly selected mode:

```
# ethtool --show-fec <netdev>
FEC parameters for <netdev>:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: Off
```

Revert back to the default Auto setting:

```
# ethtool --set-fec <netdev> encoding auto
```

Verify the setting again:

```
# ethtool --show-fec <netdev>
FEC parameters for <netdev>:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: Auto
```

Note: FEC and auto negotiation settings are persisted on the SmartNIC across reboots.

12.5 Setting Interface Breakout Mode

The following commands only work on kernel versions 4.13 and later. If your kernel is older than 4.13 or you do not have `devlink` support enabled, refer to the following section on configuring interfaces: *Configure Media Settings*.

Note: Breakout mode settings are only applicable to CX 40GbE and CX 2x40GbE SmartNICs.

Determine the card's PCI address with the `lspci` command with the correct vendor ID. The PCI vendor identifier for SmartNICs with Board Support Package (BSP) versions before 22.09 is 19ee and the specific PCI vendor identifier for SmartNICs with AMDA2XXX product codes, with a BSP version of at least 22.09, is 1da8.

For SmartNICs with a vendor ID of 19ee:

```
# lspci -bDnnd 19ee:
0000:02:00.0 Ethernet controller [0200]: Netronome Systems, Inc. Device [19ee:4000]
```

Or for SmartNICs with a vendor ID of 1da8:

```
# lspci -bDnnd 1da8:
0000:17:00.0 Ethernet controller [0200]: Corigine, Inc. Device [1da8:3800]
```

List the devices:

```
# devlink dev show
pci/0000:04:00.0
```

Split the first physical 40G port from 1x40G to 4x10G ports:

```
# devlink port split pci/0000:04:00.0/0 count 4
```

Split the second physical 40G port from 1x40G to 4x10G ports:

```
# devlink port split pci/0000:04:00.0/4 count 4
```

If the SmartNIC's port is already configured in breakout mode (it has already been split) then `devlink` will respond with an argument error. Whenever changes to the port configuration are made, the original `netdevs` associated with the port will be removed from the system.

```
# dmesg | tail
[ 5696.432306] nfp 0000:04:00.0: nfp: Port #0 config changed, unregistering.␣
↪Driver reload required before port will be operational again.
[ 6270.553902] nfp 0000:04:00.0: nfp: Port #4 config changed, unregistering.␣
↪Driver reload required before port will be operational again.
```

The driver needs to be reloaded for the changes to take effect. Older driver/SmartNIC NVRAM versions may require a system reboot for changes to take effect. The driver communicates events related to port split/unsplit in the system logs. The driver may be reloaded with the following command:

```
# rmmod nfp; modprobe nfp
```

After reloading the driver, the `netdevs` associated with the split ports will be available for use:

```
# ip link show
...
68: enp4s0np0s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
69: enp4s0np0s1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
70: enp4s0np0s2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
71: enp4s0np0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
72: enp4s0np1s0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
73: enp4s0np1s1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
74: enp4s0np1s2: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
75: enp4s0np1s3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT␣
↪group default qlen 1000
```

Note: There is an ordering constraint to splitting and unsplitting the ports on CX 2x40GbE SmartNICs. The first physical 40G port cannot be split without the second physical port also being split, hence, 1x40G + 4x10G is always invalid even if it's only intended to be a transitional mode. The driver will reject such configurations.

Breakout mode persists on the SmartNIC across reboots. To revert back to the original 2x40G ports use the unsplit subcommand.

To unsplit port 1:

```
# devlink port unsplit pci/0000:04:00.0/4
```

To unsplit port 0:

```
# devlink port unsplit pci/0000:04:00.0/0
```

The NFP drivers will again have to be reloaded (`rmmod nfp` then `modprobe nfp`) for unsplit changes in the port configuration to take effect.

12.6 Confirming Connectivity

12.6.1 Allocating IP Addresses

Under RHEL 7.5+ and CentOS 7.5+, the network configuration is managed by default using *NetworkManager*. It is recommended to disable *NetworkManager* on the NFP interfaces when using OVS-TC, as it can interfere with the TC rules that get installed on the interfaces. The easiest way to achieve this is to configure *NetworkManager* to ignore interfaces which are bound to NFP drivers. The config file for this can be created with the following script:

```
cat >/etc/NetworkManager/conf.d/nfp.conf << EOF
[keyfile]
unmanaged-devices=driver:nfp,driver:nfp_netvf,except:interface-name=ens1
EOF
systemctl restart NetworkManager
```

Verification can be done by looking at the output of `nmcli d` before and after the commands above. All the interfaces that are bound to the `nfp` or `nfp_netvf` driver, except the PF `ens1`, should now be in the `unmanaged` state.

Use `iproute2` to configure an IP on the port for a quick connectivity test. Remember to also make sure that the PF is up, `ens1` in the example below:

```
# ip address add 10.0.0.2/24 dev ens1np0
# ip link set ens1np0 up
# ip link set ens1 up
```

12.6.2 Pinging interfaces

After you have successfully assigned IP addresses to the NFP interfaces, perform a ping to another address on the same subnet to test to confirm connectivity:

```
# ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.067 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.062 ms
```

13 Basic Firmware Features

In this section `ethtool` will be used to view and configure SmartNIC interface parameters.

13.1 Summary of Features

The following table summarizes the features of the OVS-TC firmware. More detailed summaries follow hereafter.

Flow Based Features	*Flow Match Offload*
	Flow Action Offload
More Advanced Flows	*Tunnel Match Fields (General)*
	Tunnel Set Fields (General)
	Tunnel Types
	Conntrack
	QoS - Metering
	Overlay Tunnel
Configurations	*Bonding (Using Kernel Bonds)*
	Bonding (Using OVS Bonds)
	Tunnel + Bonding
	Tunnel + VLAN
	Tunnel + VLAN + Bonding
	Two Different Tunnel Configurations
	Ingress QoS
	Metering
	OpenStack OVN + XVIO Support
Other	*VFs*
	Wildcard Flows
	Ethtool Offloads
	Max MTU

continues on next page

continued from previous page

	Fallback Path for Unsupported Flows
	Port Breakout Nodes
	DPDK RTE Flow - Basic Flow Offload
	DPDK RTE Flow - VXLAN/GRE/GENEVE Tunnel Offload

13.1.1 Flow Based Features

Flow Match Offload

in_port	
Layer 2	src_mac, dst_mac
Layer 2.5	mpls, label, tos, bos
	Single VLAN: VID, TCI, PCP
	Double VLAN (QinQ): VID, TCI, PCP in both fields
Layer 3	IPv4: src, dst, proto ttl, ToS, Frag
	IPv6: src, dst, next header, hop limit, tos, frag
Layer 4	TCP: src, dst, flags
	UDP: src, dst
	SCTP: src, dst

Flow Action Offload

Layer 2	set_src, set_dst
Layer 2.5	VLAN: push, pop, set
	MPLS: push, pop, set
Layer 3	IPv4: set_src, set_dst, set_ttl, set_tos
	IPv6: set_src, set_dst, set_ttl, set_tos
Layer 4	TCP: set_sport, set_dport
	UDP: set_sport, set_dport

13.1.2 More Advanced Flows

Tunnel Match Fields (General)

- tun_ID
- Outer IPv4: src, dst
- Outer IPv6: src, dst

Tunnel Set Fields (General)

- tun_ID
- Unmasked set of outer IPv4 src/dst
- Unmasked set of outer IPv6 src/dst

Tunnel Types

- VXLAN
- GENEVE
- GENEVE + options
- NVGRE (GRE with next header Ethernet)

Conntrack

- +trk, +est flows a requirement
- extra conntrack fields: ct_zone, ct_label, ct_mark
- Protocols: IPv4: TCP, UDP
- Protocols: IPv6: TCP, UDP
- Nat

QoS - Metering

- Support meter table rate limiting shared between multiple flow rules
- Only support meter table with one band
- Only support meter table with action result of drop which is typical for a meter table
- Support meter table rate limiting base on PPS and BPS

Overlay Tunnel

- Support tunnel offload of VXLAN, GENEVE and GRE.
- Tunnel IP address can be configured on OVS bridge, PHY interface or bond interface.

13.1.3 Configurations

Bonding (Using Kernel Bonds)

- Active-backup (mode 1)
- Balance XoR (mode 2)
- hash_policy (layer3+4) or (encap3+4)
- 802.3ad (mode 4)
- Teamd, teamdctl can also be used to configure linux bonds.

Bonding (Using OVS Bonds)

- Active-backup
- Balance-slb
- Balance-tcp (but only if recirculation is turned off)

Tunnel + Bonding

- All supported types can also be used with Bonding (Linux bonds only).

Tunnel + VLAN

- A single VLAN on a tunnel outer header is supported.

Tunnel + VLAN + Bonding

- This can be combined with a single outer VLAN.

Two Different Tunnel Configurations

- IP-on-the-port: The tunnel endpoint IP is applied to the phy-port representor . The phy-port representor is NOT added to a bridge.
- IP-on-the-bridge: The tunnel endpoint IP is applied to the bridge, and the phy-port representor IS added to that bridge.

Ingress QoS

- Bits-per-second (BPS)
- Packets-per-second (PPS)
- BPS+PPS combined

Metering

- Bits-per-second (BPS)
- Packets-per-second (PPS)

OpenStack OVN + XVIO Support

- Flow tables offloading of OpenStack in the framework
- Based on OVN + XVIO

13.1.4 Other

VFs

- 55 VFs

Wildcard Flows

- 480k on CX cards
- 960k on LX cards

Ethtool Offloads

- rx/tx checksumming
- scatter gather
- TSO
- GSO
- GRO

Max MTU

- 9532

Fallback Path for Unsupported Flows

- Flows which are not supported for offloading will still traverse the system, just at a much slower rate.
- 8Q's available to handle such traffic.

Port Breakout Nodes

- 40G ports can be split into 4x10G ports
- 25G ports can be set to 10G

DPDK RTE Flow - Basic Flow Offload

- Setting of SRC/DST MAC address
- Setting of SRC/DST IPv4 address
- Setting of SRC/DST IPv6 address
- Setting of SRC/DST port
- Setting of TTL
- Setting of IPv4/IPv6 DSCP
- Push/Pop VLAN

DPDK RTE Flow - VXLAN/GRE/GENEVE Tunnel Offload

- The encap/decap action of IPv4/IPv6 VxLAN tunnel
- The encap/decap action of IPv4/IPv6 NVGRE tunnel
- The encap/decap action of IPv4/IPv6 GENEVE tunnel

13.2 View Interface Parameters

The `-k` flag can be used to view current interface configurations. For example, using a CX 1x40GbE NIC with a physical port representor `<netdev>`:

```
# ethtool -k <netdev>
Features for <netdev>:
rx-checksumming: off [fixed]
tx-checksumming: off
tx-checksum-ipv4: off [fixed]
```

(continues on next page)

(continued from previous page)

```
tx-checksum-ip-generic: off [fixed]
tx-checksum-ipv6: off [fixed]
tx-checksum-fcoe-crc: off [fixed]
tx-checksum-sctp: off [fixed]
scatter-gather: off
tx-scatter-gather: off [fixed]
tx-scatter-gather-fraglist: off [fixed]
tcp-segmentation-offload: off
tx-tcp-segmentation: off [fixed]
tx-tcp-ecn-segmentation: off [fixed]
tx-tcp6-segmentation: off [fixed]
tx-tcp-mangleid-segmentation: off [fixed]
udp-fragmentation-offload: off [fixed]
generic-segmentation-offload: off [requested on]
generic-receive-offload: on
large-receive-offload: off [fixed]
rx-vlan-offload: off [fixed]
tx-vlan-offload: off [fixed]
ntuple-filters: off [fixed]
receive-hashing: off [fixed]
highdma: off [fixed]
rx-vlan-filter: off [fixed]
vlan-challenged: off [fixed]
tx-lockless: off [fixed]
netns-local: off [fixed]
tx-gso-robust: off [fixed]
tx-fcoe-segmentation: off [fixed]
tx-gre-segmentation: off [fixed]
tx-ipip-segmentation: off [fixed]
tx-sit-segmentation: off [fixed]
tx-udp_tnl-segmentation: off [fixed]
fcoe-mtu: off [fixed]
tx-nocache-copy: off
loopback: off [fixed]
rx-fcs: off [fixed]
rx-all: off [fixed]
tx-vlan-stag-hw-insert: off [fixed]
rx-vlan-stag-hw-parse: off [fixed]
rx-vlan-stag-filter: off [fixed]
busy-poll: off [fixed]
tx-gre-csum-segmentation: off [fixed]
tx-udp_tnl-csum-segmentation: off [fixed]
tx-gso-partial: off [fixed]
tx-sctp-segmentation: off [fixed]
l2-fwd-offload: off [fixed]
hw-tc-offload: on
rx-udp_tunnel-port-offload: off [fixed]
```

Note: Replace `<netdev>` with the machine's specific interface associated with the SmartNIC's PF, which is expected to be something like `enp4s0` or `enp4s0np0`.

13.3 Configuring Interface Settings

Unless otherwise stated, changing the interface settings detailed below will not require reloading of the NFP drivers for changes to take effect, unlike the interface breakouts described in *Configuring Interface Media Mode*. If the interface has more than one physical port, changes must be applied to the physical function `<netdev>` and those settings will reflect on both ports. Unlike the basic CoreNIC firmware, each physical port on the interface cannot be configured independently and attempting to do so will produce an error.

13.3.1 Receive Checksum Offload

When enabled, checksum calculation and error checking comparison for received packets is offloaded to the NFP SmartNIC's flow processor rather than the host CPU.

To enable receive checksum offload:

```
# ethtool -K <netdev> rx on
```

To disable receive checksum offload:

```
# ethtool -K <netdev> rx off
```

13.3.2 Transmit Checksum Offload

When enabled, checksum calculation for outgoing packets is offloaded to the NFP SmartNIC's flow processor rather than the host's CPU.

To enable transmit checksum offload:

```
# ethtool -K <netdev> tx on
```

To disable transmit checksum offload:

```
# ethtool -K <netdev> tx off
```

13.3.3 Scatter/Gather

When enabled the NFP will use scatter/gather I/O, also known as Vectored I/O, which allows a single procedure call to sequentially read data from multiple buffers and write it to a single data stream. Only changes to the scatter-gather interface settings (from `on` to `off` or `off` to `on`) will produce a terminal output as shown below:

```
# ethtool -K <netdev> sg on
Actual changes:
scatter-gather: on
tx-scatter-gather: on
generic-segmentation-offload: on

# ethtool -K <netdev> sg off
Actual changes:
scatter-gather: off
tx-scatter-gather: off
generic-segmentation-offload: off
```

13.3.4 TCP Segmentation Offload (TSO)

When enabled, this parameter causes all functions related to the segmentation of TCP packets at egress to be offloaded to the NFP.

To enable TCP segmentation offload:

```
# ethtool -K <netdev> tso on
```

To disable TCP segmentation offload:

```
# ethtool -K <netdev> tso off
```

13.3.5 Generic Segmentation Offload (GSO)

This parameter offloads segmentation for transport layer protocol data units other than segments and datagrams for TCP/UDP respectively to the NFP. GSO operates at packet egress.

To enable generic segmentation offload:

```
# ethtool -K <netdev> gso on
```

To disable generic segmentation offload:

```
# ethtool -K <netdev> gso off
```

13.3.6 Generic Receive Offload (GRO)

This parameter enables software implementation of Large Receive Offload (LRO), which aggregates multiple packets at ingress into a large buffer before they are passed higher up the networking stack.

To enable generic receive offload:

```
# ethtool -K <netdev> gro on
```

To disable generic receive offload:

```
# ethtool -K <netdev> gro off
```

Note: Take note that scripts that use `ethtool -i <netdev>` to get bus-info will not work on representors as this information is not populated for representor devices.

14 Using Open vSwitch

14.1 Running Open vSwitch

14.1.1 RHEL and CentOS

The first step is to start Open vSwitch by using the following command:

```
# systemctl start openvswitch
```

View the status of Open vSwitch with the command below:

```
# systemctl status openvswitch
● openvswitch.service - Open vSwitch
   Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; enabled; vendor␣
↪preset: disabled)
   Active: active (exited) since Tue 2022-06-28 22:25:16 SAST; 9h ago
  Process: 1196 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
 Main PID: 1196 (code=exited, status=0/SUCCESS)
    Tasks: 0 (limit: 151960)
   Memory: 0B
   CGroup: /system.slice/openvswitch.service

Jun 28 22:25:16 nala systemd[1]: Starting Open vSwitch...
Jun 28 22:25:16 nala systemd[1]: Started Open vSwitch.
```

The *openvswitch* service controls the *ovsdb-server* and *ovs-vswitchd* services. Their statuses can also be checked by running the commands below:

```
# systemctl status ovsdb-server
● ovsdb-server.service - Open vSwitch Database Unit
   Loaded: loaded (/usr/lib/systemd/system/ovsdb-server.service; static; vendor␣
↪preset: disabled)
   Active: active (running) since Tue 2022-06-28 22:25:15 SAST; 9h ago
  Process: 995 ExecStart=/usr/share/openvswitch/scripts/ovs-ctl --no-ovs-vswitchd -
↪-no-monitor --system-id=random ${OVS_USER_OPT} start $OPTIONS (code=exited>
  Process: 992 ExecStartPre=/bin/sh -c if [ "$${OVS_USER_ID/:*/}" != "root" ];␣
↪then /usr/bin/echo "OVS_USER_OPT=--ovs-user=${OVS_USER_ID}" >> /run/openvswitc>
  Process: 978 ExecStartPre=/bin/sh -c /usr/bin/echo "OVS_USER_ID=${OVS_USER_ID}" >
↪ /run/openvswitch.useropts (code=exited, status=0/SUCCESS)
  Process: 917 ExecStartPre=/usr/bin/chown ${OVS_USER_ID} /var/run/openvswitch /
↪var/log/openvswitch (code=exited, status=1/FAILURE)
  Process: 914 ExecStartPre=/usr/bin/rm -f /run/openvswitch.useropts (code=exited,␣
↪status=0/SUCCESS)
```

(continues on next page)

(continued from previous page)

```
 Main PID: 1094 (ovsdb-server)
    Tasks: 1 (limit: 151960)
   Memory: 37.4M
   CGroup: /system.slice/ovsdb-server.service
           └─1094 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer -
↪vsyslog:err -vfile:info --remote=punix:/var/run/openvswitch/db.sock --private-
↪key=db:>

Jun 28 22:25:12 nala systemd[1]: Starting Open vSwitch Database Unit...
Jun 28 22:25:13 nala chown[917]: /usr/bin/chown: cannot access '/var/run/
↪openvswitch': No such file or directory
Jun 28 22:25:15 nala ovs-ctl[995]: Starting ovsdb-server [  OK  ]
Jun 28 22:25:15 nala ovs-vsctl[1100]: ovs|00001|vsctl|INFO|Called as ovs-vsctl --
↪no-wait -- init -- set Open_vSwitch . db-version=8.2.0
Jun 28 22:25:15 nala ovs-vsctl[1112]: ovs|00001|vsctl|INFO|Called as ovs-vsctl --
↪no-wait set Open_vSwitch . ovs-version=2.15.6 "external-ids:system-id=\"8971>
Jun 28 22:25:15 nala ovs-ctl[995]: Configuring Open vSwitch system IDs [  OK  ]
Jun 28 22:25:15 nala ovs-vsctl[1118]: ovs|00001|vsctl|INFO|Called as ovs-vsctl --
↪no-wait add Open_vSwitch . external-ids hostname=nala
Jun 28 22:25:15 nala ovs-ctl[995]: Enabling remote OVSDB managers [  OK  ]
Jun 28 22:25:15 nala systemd[1]: Started Open vSwitch Database Unit.
...
```

```
# systemctl status ovs-vswitchd
● ovs-vswitchd.service - Open vSwitch Forwarding Unit
   Loaded: loaded (/usr/lib/systemd/system/ovs-vswitchd.service; static; vendor␣
↪preset: disabled)
   Active: active (running) since Tue 2022-06-28 22:25:16 SAST; 9h ago
  Process: 1135 ExecStart=/usr/share/openvswitch/scripts/ovs-ctl --no-ovsdb-server␣
↪--no-monitor --system-id=random ${OVS_USER_OPT} start $OPTIONS (code=exite>
  Process: 1133 ExecStartPre=/usr/bin/chmod 0775 /dev/hugepages (code=exited,␣
↪status=0/SUCCESS)
  Process: 1131 ExecStartPre=/bin/sh -c /usr/bin/chown :$${OVS_USER_ID##*:} /dev/
↪hugepages (code=exited, status=0/SUCCESS)
 Main PID: 1187 (ovs-vswitchd)
    Tasks: 1 (limit: 151960)
   Memory: 113.2M
   CGroup: /system.slice/ovs-vswitchd.service
           └─1187 ovs-vswitchd unix:/var/run/openvswitch/db.sock -vconsole:emer -
↪vsyslog:err -vfile:info --mlockall --user openvswitch:hugetlbfs --no-chdir ->

Jun 28 22:25:15 nala systemd[1]: Starting Open vSwitch Forwarding Unit...
Jun 28 22:25:16 nala ovs-ctl[1168]: Inserting openvswitch module [  OK  ]
Jun 28 22:25:16 nala ovs-ctl[1135]: Starting ovs-vswitchd [  OK  ]
Jun 28 22:25:16 nala ovs-vsctl[1195]: ovs|00001|vsctl|INFO|Called as ovs-vsctl --
↪no-wait add Open_vSwitch . external-ids hostname=nala
Jun 28 22:25:16 nala ovs-ctl[1135]: Enabling remote OVSDB managers [  OK  ]
Jun 28 22:25:16 nala systemd[1]: Started Open vSwitch Forwarding Unit.
...
```

Open vSwitch can be enabled to run on reboot. This is done below:

```
# systemctl enable openvswitch
```

14.1.2 Ubuntu

The first step is to start Open vSwitch by using the following command:

```
# systemctl start openvswitch-switch
```

View the status of Open vSwitch with the command below:

```
# systemctl status openvswitch-switch
● openvswitch-switch.service - Open vSwitch
   Loaded: loaded (/lib/systemd/system/openvswitch-switch.service; enabled; vend
   Active: active (exited) since Wed 2018-05-09 08:35:44 UTC; 20s ago
 Main PID: 1824 (code=exited, status=0/SUCCESS)
    Tasks: 0 (limit: 1153)
   CGroup: /system.slice/openvswitch-switch.service
```

The *openvswitch-vswitch* service controls the *ovsdb-server* and *ovs-vswitchd* services. Their statuses can also be checked by running the commands below:

```
# systemctl status ovsdb-server
● ovsdb-server.service - Open vSwitch Database Unit
   Loaded: loaded (/lib/systemd/system/ovsdb-server.service; static; vendor pres
   Active: active (running) since Wed 2018-05-09 08:35:44 UTC; 1min 38s ago
    Tasks: 1 (limit: 1153)
   CGroup: /system.slice/ovsdb-server.service
              └─1749 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer -vsyslog:


# systemctl status ovs-vswitchd
● ovs-vswitchd.service - Open vSwitch Forwarding Unit
   Loaded: loaded (/lib/systemd/system/ovs-vswitchd.service; static; vendor pres
   Active: active (running) since Wed 2018-05-09 08:35:44 UTC; 2min 6s ago
 Main PID: 1813 (ovs-vswitchd)
    Tasks: 1 (limit: 1153)
   CGroup: /system.slice/ovs-vswitchd.service
           └─1813 ovs-vswitchd unix:/var/run/openvswitch/db.sock -vconsole:emer
```

Open vSwitch can be enabled to run on reboot. This is done below:

```
# systemctl enable openvswitch-switch
Synchronizing state of openvswitch-switch.service with SysV service script with /
↪lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable openvswitch-switch
```

14.2 Configuring Open vSwitch Hardware Offload

To enable TC offloading in Open vSwitch, the `hw-tc-offload` flag for the representors of any ports that will send or receive offloaded traffic must be set to true. Unlike interface settings described in *Configuring Interface Settings* `hw-tc-offload` flags must be set for each physical port representor. Hardware TC offload is enabled by default and can be verified for each port using `ethtool`. Note that the PF interface won't show the `hw-tc-offload` flag being set by default. For example:

```
# ethtool -k <netdev> | grep hw-tc-offload
hw-tc-offload: on
```

Note: Replace `<netdev>` with the machine's specific interface associated with the SmartNIC's PF, which is expected to be something like `ens4`.

The setting may be toggled for each port independently between on and off using `ethtool`:

```
# ethtool -K <netdev> hw-tc-offload on
```

Note: Hardware offload changes won't persist across reboots. The default setting for TC offloads when using the `flower` firmware is `on`.

The Open vSwitch hardware offload is configured as follows:

```
# ovs-vsctl set Open_vSwitch . other_config:hw-offload=true other_config:tc-
↪policy=none
```

This change will persist across reboots. But, in the absence of a reboot, Open vSwitch must be restarted:

In RHEL and CentOS this is performed by the command:

```
# systemctl restart openvswitch
```

In Ubuntu, the following command is used instead:

```
# systemctl restart openvswitch-switch
```

14.3 Open vSwitch Hardware Offload Example

Create an Open vSwitch bridge and add two interfaces: the representors of the first physical port and the VF. Please refer to section *SmartNIC Netdev Interfaces* for information on `netdevs` of the SmartNICs and *Configuring SR-IOV* for creating VFs associated with a physical interface. The following example requires at least one VF representor (in this case `eth1`) associated with the PF `netdev`.

Create an Open vSwitch bridge:

```
# ovs-vsctl add-br br0
```

Add representor `netdev` for the first physical port to the bridge:

```
# ovs-vsctl add-port br0 <netdev>
```

Add the representor `netdev` of the first VF to bridge:

```
# ovs-vsctl add-port br0 eth1
```

The `ovs-vsctl show` command can be used to verify the config of the bridge, and the kernel datapath can be verified with `ovs-dpctl show`:

```
# ovs-vsctl show
5e9b8d4b-4a29-41af-92f1-3d9f161aa176
    Bridge "br0"
        Port "br0"
            Interface "br0"
                type: internal
        Port "eth1"
            Interface "eth1"
        Port "ens4"
            Interface "ens4"
    ovs_version: "2.15.6"

# ovs-dpctl show
system@ovs-system:
  lookups: hit:19 missed:14 lost:0
  flows: 14
  masks: hit:84 total:5 hit/pkt:2.55
  port 0: ovs-system (internal)
  port 1: br0 (internal)
  port 2: enp4s0np0
  port 3: eth1
```

Packets should now be able to flow between the VF and the external port. The view of Open vSwitch for offloaded and non-offloaded flows can be seen listed using `ovs-appctl`. The port numbers used for `in_port` and the (output) actions correspond to those listed by `ovs-appctl` show as shown above.

To view the offloaded datapath flows, use the command below:

```
# ovs-appctl dpctl/dump-flows type=offloaded
in_port(2),eth(src=00:15:4d:0e:08:a7,dst=66:11:3e:c9:cf:2f),eth_type(0x0806),␣
↪packets:2, bytes:92, used:187.890s, actions:3
in_port(2),eth(src=00:15:4d:0e:08:a7,dst=66:11:3e:c9:cf:2f),eth_type(0x0800),␣
↪packets:9, bytes:882, used:188.860s, actions:3
...
```

To view the non-offloaded datapath flows, use the command below:

```
# ovs-appctl dpctl/dump-flows type=ovs
recirc_id(0),in_port(3),eth(src=66:11:3e:c9:cf:2f,dst=33:33:ff:c9:cf:2f),eth_
↪type(0x86dd),ipv6(frag=no), packets:0, bytes:0, used:never, actions:1,2
recirc_id(0),in_port(3),eth(src=66:11:3e:c9:cf:2f,dst=33:33:00:00:00:02),eth_
↪type(0x86dd),ipv6(frag=no), packets:2, bytes:140, used:1399.137s, actions:1,2
...
```

To view both offloaded and non-offloaded datapath flows, use the command below:

```
# ovs-appctl dpctl/dump-flows
in_port(2),eth(src=00:15:4d:0e:08:a7,dst=66:11:3e:c9:cf:2f),eth_type(0x0806),␣
↪packets:2, bytes:92, used:187.890s, actions:3
in_port(2),eth(src=00:15:4d:0e:08:a7,dst=66:11:3e:c9:cf:2f),eth_type(0x0800),␣
↪packets:9, bytes:882, used:188.860s, actions:3
...
recirc_id(0),in_port(3),eth(src=66:11:3e:c9:cf:2f,dst=33:33:ff:c9:cf:2f),eth_
↪type(0x86dd),ipv6(frag=no), packets:0, bytes:0, used:never, actions:1,2
recirc_id(0),in_port(3),eth(src=66:11:3e:c9:cf:2f,dst=33:33:00:00:00:02),eth_
↪type(0x86dd),ipv6(frag=no), packets:2, bytes:140, used:1399.137s, actions:1,2
...
```

Note: `type=offloaded` is just an indication that a flow is handled by the TC datapath. This does not guarantee that it has been offloaded to the SmartNIC, the TC commands shown next provides a much better indication.

The non-offloaded flows are present in the Open vSwitch kernel datapath. The offloaded flows are present in hardware, and are configured by Open vSwitch via the Kernel's TC subsystem. The kernel's view of these flows may be observed using the `tc` command:

```
# tc -s filter show ingress dev <netdev>
filter protocol arp pref 1 flower
filter protocol arp pref 1 flower handle 0x1
  dst_mac 66:11:3e:c9:cf:2f
  src_mac 00:15:4d:0e:08:a7
  eth_type arp
  not_in_hw
      action order 1: mirred (Egress Redirect to device eth1) stolen
      index 1 ref 1 bind 1 installed 409 sec used 187 sec
      Action statistics:
      Sent 92 bytes 2 pkt (dropped 0, overlimits 0 requeues 0)
      backlog 0b 0p requeues 0
      cookie len 16 e053c4819648461a

filter protocol ip pref 2 flower
filter protocol ip pref 2 flower handle 0x1
  dst_mac 66:11:3e:c9:cf:2f
```

(continues on next page)

(continued from previous page)

```
  src_mac 00:15:4d:0e:08:a7
  eth_type ipv4
  in_hw
      action order 1: mirred (Egress Redirect to device eth1) stolen
      index 4 ref 1 bind 1 installed 409 sec used 188 sec
      Action statistics:
      Sent 882 bytes 9 pkt (dropped 0, overlimits 0 requeues 0)
      backlog 0b 0p requeues 0
      cookie len 16 b68ca7de9c465000

# tc -s filter show ingress dev eth1
filter protocol arp pref 1 flower
filter protocol arp pref 1 flower handle 0x1
  dst_mac 00:15:4d:0e:08:a7
  src_mac 66:11:3e:c9:cf:2f
  eth_type arp
  not_in_hw
      action order 1: mirred (Egress Redirect to device enp4s0np0) stolen
      index 2 ref 1 bind 1 installed 409 sec used 187 sec
      Action statistics:
      Sent 56 bytes 2 pkt (dropped 0, overlimits 0 requeues 0)
      backlog 0b 0p requeues 0
      cookie len 16 5049f238734ef962

filter protocol ip pref 2 flower
filter protocol ip pref 2 flower handle 0x1
  dst_mac 00:15:4d:0e:08:a7
  src_mac 66:11:3e:c9:cf:2f
  eth_type ipv4
  in_hw
      action order 1: mirred (Egress Redirect to device enp4s0np0) stolen
      index 3 ref 1 bind 1 installed 409 sec used 188 sec
      Action statistics:
      Sent 882 bytes 9 pkt (dropped 0, overlimits 0 requeues 0)
      backlog 0b 0p requeues 0
      cookie len 16 3dae846e6b41a778
```

15 Appendix A: Corigine Repositories

All the software mentioned in this document can be obtained via the official Corigine repositories. Please find instructions below on how to enable access to the aforementioned repositories from your respective Linux distributions.

15.1 Importing GPG-Key

For RHEL and CentOS, add the Corigine GPG-key:

```
# rpm --import https://download.corigine.com.cn/public/Corigine.pub
```

For Ubuntu based systems, add the Corigine GPG-key:

```
# curl -fsSLo /usr/share/keyrings/corigine-archive-keyring.gpg \
https://download.corigine.com.cn/public/Corigine.gpg
```

15.2 Configuring Repositories

For RHEL 7 and CentOS 7, the RPM repository can be added:

```
# yum-config-manager --add-repo \
https://download.corigine.com.cn/public/corigine.repo
```

For RHEL 8+ and CentOS 8+, the RPM repository can be added:

```
# dnf config-manager --add-repo \
https://download.corigine.com.cn/public/corigine.repo
```

For Ubuntu based systems:

```
# mkdir -p /etc/apt/sources.list.d
# KEY=/usr/share/keyrings/corigine-archive-keyring.gpg
# REPOLINK=https://download.corigine.com.cn/public/apt
# OUPUTPATH=/etc/apt/sources.list.d/corigine.list
# echo "deb [arch=all signed-by=${KEY}] ${REPOLINK} stable main" > ${OUPUTPATH}
# apt-get update
```

16 Appendix B: Red Hat Repositories

TC offload is only available in Open vSwitch version 2.8, with additional offloads enabled thereafter. The standard Red Hat Subscription only enables Open vSwitch version 2.5. For this reason, an additional subscription may be required to enable repositories that contain a newer version of Open vSwitch. Please consult with Red Hat directly to determine your subscription needs. More information is available at the official Red Hat documentation page.

Note: The Red Hat documentation with regards to enabling the specific repositories is regarded to be authoritative. The steps below are for illustrative purposes only.

Register the system with `subscription-manager`:

```
# subscription-manager register
```

List all available pools:

```
# subscription-manager list --all --available
```

Identify the IDs of the license pools that provide the following products:

- Red Hat Enterprise Linux
- Red Hat Enterprise Linux Fast Datapath

This can be done by using the `--matches` flag:

```
# subscription-manager list --available --matches="Red Hat Enterprise
Linux Fast Datapath"
```

Attach the system to these pools (by using the correct license pool IDs):

```
# subscription-manager attach --pool=${RHEL_PACKAGE_POOL_ID}
```

Enable the Fast Datapath repository for the relevant version of RHEL:

RHEL 7 and CentOS 7:

```
# subscription-manager repos --enable rhel-7-fast-datapath-rpms
```

RHEL 8 and CentOS 8:

```
# subscription-manager repos --enable fast-datapath-for-rhel-8-x86_64-rpms
```

17 Appendix C: Installing the Out-of-Tree NFP Driver

The NFP driver can be installed via the Corigine repository, or built from source, depending on your requirements.

Note: The Out-of-Tree driver currently does not provide support for TC firmware on RHEL/CentOS 7.

17.1 Install Driver via Corigine Repository

Please refer to *Appendix A: Corigine Repositories* on how to configure the Corigine repository applicable to your distribution. When the repository has been successfully added, install the NFP driver package using the commands below.

17.1.1 RHEL 8 and CentOS 8

When installing the NFP DKMS driver package, DKMS is required as a dependency. On RHEL based systems, DKMS is provided in the EPEL repository. If this is not installed, it must first be done before installing the NFP driver package. The EPEL repository can be installed using:

```
# dnf install epel-release
```

Ensure that the correct kernel-development package is installed that matches the current kernel version. The following command will check the kernel-devel version and, if needed, install the correct kernel-devel package:

```
# dnf install kernel-devel-$(uname -r)
```

Installing the driver from the Corigine repository should automatically install all dependencies:

```
# dnf search agilio-nfp-driver-dkms
# dnf install agilio-nfp-driver-dkms
```

17.1.2 Ubuntu

```
# apt-cache search agilio-nfp-driver-dkms
# apt-get install agilio-nfp-driver-dkms
```

17.1.3 Kernel Changes

Take note that installing the DKMS driver will only install it for the currently running kernel. When you upgrade the installed kernel it may not automatically update the `nfp` module to use the version in the DKMS package.In kernel versions older than v4.16, the `MODULE_VERSION` parameter of the in-tree module was not set, which causes DKMS to pick the module with the highest `srcversion` hash (https://github.com/dell/dkms/issues/14). The work around for this is to add a `--force` flag to the DKMS install in the package install step, but this will not trigger on a kernel upgrade. To work around this issue, boot into the new kernel and then re-install the `agilio-nfp-driver-dkms` package.

This should not be a problem when upgrading from kernels v4.16 and newer as the `MODULE_VERSION` has been added and the DKMS version check should work properly. It's not possible to determine which `nfp.ko` file was loaded by only relying on information provided by the kernel. However, it's possible to confirm that the binary signature of a file on disk and the module loaded in memory is the same.

To confirm that the module in memory is the same as the file on disk, compare the `srcversion` tag. The in-memory module's tag is at `/sys/module/nfp/srcversion`. The default on-disk version can be queried with `modinfo`:

```
# cat /sys/module/nfp/srcversion # In-memory module
# modinfo nfp | grep "^srcversion:" # On-disk module
```

If these tags are in sync, the filename of the module provided by a `modinfo` query will identify the origin of the module:

```
# modinfo nfp | grep "^filename:"
```

If these tags are not in sync, there are likely conflicting copies of the module on the system: the `initramfs` may be out of sync or the module dependencies may be inconsistent.

The in-tree kernel module is usually located at the following path (please note, this module may be compressed with a `.xz` extension):

```
/lib/modules/$(uname -r)/kernel/drivers/net/ethernet/netronome/nfp/nfp.ko
```

The DKMS module is usually located at the following path:

```
/lib/modules/$(uname -r)/updates/dkms/nfp.ko
```

To ensure that the out-of-tree driver is correctly loaded instead of the in-tree module, the following commands can be run:

```
# mkdir -p /etc/depmod.d
# echo "override nfp * extra" > /etc/depmod.d/netronome.conf
```

(continues on next page)

(continued from previous page)

```
# depmod -a
# rmmod nfp; modprobe nfp
# update-initramfs -u
```

17.2 Building from Source

Driver sources for Corigine Network Flow Processor devices, including the NFP-4000 and NFP-6000 models can be found at: https://github.com/Corigine/nfp-drv-kmods

17.2.1 RHEL 8 and CentOS 8 Dependencies

```
# dnf install -y kernel-devel-$(uname -r)
# dnf groupinstall -y "Development Tools"
```

17.2.2 Ubuntu Dependencies

```
# apt-get update
# apt-get install -y linux-headers-$(uname -r) build-essential libelf-dev
```

17.2.3 Clone, Build and Install

```
# git clone https://github.com/Corigine/nfp-drv-kmods.git
# cd nfp-drv-kmods
# make
# make install
# depmod -a
```

18 Appendix D: Working with Board Support Package

The Corigine Board Support Package (BSP) provides infrastructure software and a development environment for managing NFP based platforms.

18.1 Install Software from Corigine Repository

Please refer to *Appendix A: Corigine Repositories* on how to configure the Corigine repository applicable to your distribution. When the repository has been successfully added, install the BSP package using the commands below.

RHEL 7 and CentOS 7:

```
# yum list available | grep nfp-bsp
# yum install nfp-bsp
# reboot
```

RHEL 8 and CentOS 8:

```
# dnf list available | grep nfp-bsp
# dnf install nfp-bsp
# reboot
```

Ubuntu:

```
# apt-cache search nfp-bsp
# apt-get install nfp-bsp
```

18.2 Install Software from DEB/RPM Package

18.2.1 Obtain Software

The latest BSP packages can be obtained at the downloads area of the Corigine Support site (https://www.corigine.com/DPUDownload.html).

18.2.2 Install the Prerequisite Dependencies

RHEL and CentOS Dependencies

The libftdi package is required to install BSP software, it can be installed from the EPEL repository. Install the EPEL repository by running:

```
# yum install -y epel-release
```

Then install the libftdi package by running:

```
# yum install -y libftdi
```

Ubuntu Dependencies

To install the BSP package dependencies on Ubuntu, run:

```
# apt-get install -y libjansson4 libftdi
```

18.2.3 NFP BSP Package

Install the NFP BSP package provided by Corigine Support.

RHEL 7 and CentOS 7 Install:

```
# yum install -y nfp-bsp*.rpm
```

RHEL 8 and CentOS 8 Install:

```
# dnf install -y nfp-bsp*.rpm
```

Ubuntu Install:

```
# dpkg -i nfp-bsp*.deb
```

18.3 Using BSP Tools

18.3.1 Enable CPP Access

The NFP has an internal Command Push/Pull (CPP) bus that allows debug access to the SmartNIC internals. CPP access allows user space tools raw access to chip internals and is required to enable the use of most BSP tools. Only the out-of-tree (OOT) driver allows CPP access.

Follow the steps from *Install Driver via Corigine Repository* to install the OOT NFP driver. After the `nfp` module has been built, load the driver with CPP access:

```
# depmod -a
# rmmod nfp
# modprobe nfp nfp_dev_cpp=1
```

To persist this option across reboots, several options are available; the distribution specific documentation, which can be found at RHEL, CentOS and Ubuntu, will detail that process more thoroughly. Care must be taken that the settings are also applied to any initramfs images generated.

18.3.2 Configure Media Settings

Alternatively to the process described in *Configuring Interface Media Mode*, BSP tools can be used to configure the port speed of the SmartNIC using the following commands. Note, a reboot is still required for changes to take effect.

CX 2x25GbE - AMDA0099

To set the port speed of the CX 2x25GbE, the following commands can be used:

Set port 0 and port 1 to 10G mode:

```
# nfp-media phy1=10G phy0=10G
```

Set port 1 to 25G mode:

```
# nfp-media phy1=25G+
```

To change the FEC settings of the 2x25GbE, the following commands can be used:

```
# nfp-media --set-aneg=phy0=[S|A|I|C|F] --set-fec=phy0=[A|F|R|N]
```

Where the parameters for each argument are:

`--set-aneg=`:

S

search - Search through supported modes until link is found. Only one side should be doing this. It may result in a mode that can have physical layer errors depending on SFP type and what the other end wants. Long DAC cables with no FEC WILL have physical layer errors.

A

auto - Automatically choose mode based on speed and SFP type.

C

consortium - Consortium 25G auto-negotiation with link training.

I

IEEE - IEEE 10G or 25G auto-negotiation with link training.

F

forced - Mode is forced with no auto-negotiation or link training.

`--set-fec=`:

A

auto - Automatically choose FEC based on speed and SFP type.

F

Firecode - BASE-R Firecode FEC compatible with 10G.

R

Reed-Solomon - Reed-Solomon FEC new for 25G.

N

none - No FEC is used.

CX 1x40GbE - AMDA0081

Set port 0 to 40G mode:

```
# nfp-media phy0=40G
```

Set port 0 to 4x10G fanout mode:

```
# nfp-media phy0=4x10G
```

CX 2x40GbE - AMDA0097

Set port 0 and port 1 to 40G mode:

```
# nfp-media phy0=40G phy1=40G
```

Set port 0 to 4x10G fanout mode:

```
# nfp-media phy0=4x10G
```

For mixed configuration the highest port must be in 40G mode e.g.:

```
# nfp-media phy0=4x10G phy1=40G
```

19 Appendix E: Upgrading the Kernel

The minimum recommended Linux distribution versions are those provided in supported releases of distributions. As a guide they are as follows:

Operating System	Kernel Version
CentOS 7.6	3.10.0-957
CentOS 8.0	4.18
Ubuntu 18.04 LTS	4.15

19.1 RHEL

Only kernel packages released by Red Hat which are installable as part of the distribution installation and upgrade procedure are supported.

19.2 CentOS

The CentOS package installer yum will manage an update to the supported kernel version. The command `yum install kernel-${VERSION}` updates the kernel for CentOS. First search for available kernel packages then install the desired release:

```
# yum list --showduplicates kernel

kernel.x86_64       3.10.0-862.el7          base
kernel.x86_64       3.10.0-862.2.3.el7      updates
kernel.x86_64       3.10.0-862.3.2.el7      updates

# yum install kernel-3.10.0-862.el7
```

19.3 Ubuntu

If desired, alternative kernels may be installed. For example, at the time of writing, v4.18 is the newest stable kernel.

19.3.1 Acquire packages

```
# BASE=http://kernel.ubuntu.com/~kernel-ppa/mainline/v4.18/
# HEADERS=linux-headers-4.18.0-041800
# IMAGE=linux-image-unsigned-4.18.0-041800
# MODULES=linux-modules-4.18.0-041800-generic
# wget \
    $BASE/${HEADERS}_4.18.0-041800.201808122131_all.deb \
    $BASE/${HEADERS}-generic_4.18.0-041800.201808122131_amd64.deb \
    $BASE/${IMAGE}-generic_4.18.0-041800.201808122131_amd64.deb \
    $BASE/${MODULES}_4.18.0-041800.201808122131_amd64.deb
```

19.3.2 Install packages

```
# HEADERS=linux-headers-4.18.0-041800
# IMAGE=linux-image-unsigned-4.18.0-041800-generic
# MODULES=linux-modules-4.18.0-041800-generic
# dpkg -i \
    ${HEADERS}_4.18.0-041800.201808122131_all.deb \
    ${HEADERS}-generic_4.18.0-041800.201808122131_amd64.deb \
    ${IMAGE}_4.18.0-041800.201808122131_amd64.deb \
    ${MODULES}_4.18.0-041800.201808122131_amd64.deb
```

20 Appendix F: Updating Kernel Boot Parameters

Note: In order to enable VFs to be bound to the `vfio-pci` driver such that they may be utilized by VMs, IOMMU must be enabled in both the BIOS of the host machines, as well as the kernel.

20.1 RHEL and CentOS Grub Config

```
# grubby --update-kernel=ALL --args="intel_iommu=on"
# reboot
```

20.2 Ubuntu Grub Config

```
# sed -i \
's/#*GRUB_CMDLINE_LINUX_DEFAULT.*/GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on"/g' \
/etc/default/grub
# update-grub2
# reboot
```

21 Appendix G: Upgrading TC Firmware

The preferred method of installing and upgrading Agilio firmware is via the distribution repositories. The minimum recommended versions are those provided in GA releases of distributions. As a guide they are as follows:

Operating System	Firmware package version
CentOS 7.6	`20180911-69.git85c5d90.el7`
CentOS 8.0	`20190111-92.gitd9fb2ee6.el8`
Ubuntu 18.04 LTS	`1.173`

Corigine provides firmware packages with newer features as out-of-tree repositories. The corresponding installation packages can be obtained from Corigine Support if access to the repositories is not available. (smartnic-support@corigine.com).

21.1 Installing Updated TC Firmware via the Corigine Repository

Please refer to *Appendix A: Corigine Repositories* on how to configure the Corigine repository applicable to your distribution. When the repository has been successfully added, install the *agilio-flower-app-firmware* package using the commands below.

In RHEL and CentOS:

```
# yum install agilio-flower-app-firmware
```

In Ubuntu 18.04 LTS:

```
# apt-get install agilio-flower-app-firmware
```

21.2 Installing Updated TC Firmware from Package Installations

The latest firmware can be obtained at the downloads area of the Corigine Support site (https://www.corigine.com/DPUdownload.html). Install the packages provided by Corigine Support using the commands below.

In RHEL and CentOS:

```
# yum install -y agilio-flower-app-firmware-*.rpm
```

In Ubuntu 18.04 LTS:

```
# dpkg -i agilio-flower-app-firmware-*.deb
```

21.3 Select Updated TC Firmware

Once installed, the updated TC firmware should be selected using the script described in section *Selecting the TC Offload Firmware*. The script should be called `agilio-tc-fw-select.sh`. To select the updated TC firmware, it should be called with *flower-next* as its last parameter:

```
# ./agilio-tc-fw-select.sh <netdev> scan flower-next
```

Note: Replace `<netdev>` with the machine's specific interface associated with the SmartNIC's physical port, which is expected to be something like ens4np0.

Once selected, the driver should be reloaded to use the new firmware:

```
# rmmod nfp; modprobe nfp
```

22 Appendix H: Offloadable Flows

Flows may be offloaded to hardware if they meet the criteria described in this section.

Note: The maximum number of flows that can be offloaded in RHEL 7.5/7.6 and Ubuntu 18.04 is 128k. This has been increased to 480k in kernel 4.20 and has been backported to the 4.18-based kernel provided by RHEL 8.0.

22.1 Matches

A flow may be offloaded if it matches only on the following fields:

Meta-data	Input Port
Layer 2	Ethernet: Type, Addresses
VLAN:	Outermost ID, Priority
Layer 3	IPv4: Addresses, Protocol, TTL, TOS, Frag IPv6: Addresses, Protocol, Hop Limit, TOS, Frag
Layer 4	TCP: Ports, Flags UDP: Ports SCTP: Ports
Tunnel	ID IPv4: Outer Address UDP: Outer Destination Port

22.2 Actions

A flow may be offloaded if:

1. The input port of the flow is:
    a. A physical port or VF on an Agilio SmartNIC or;
    b. A supported tunnel vport whose ingress packets are received on a physical port on an Agilio SmartNIC and whose egress action is to a VF port on an Agilio SmartNIC.
2. If present, the output actions output to:
    a. A physical port or VF on the same Agilio SmartNIC as the input port;
    b. A tunnel vport whose egress packets are sent on a physical port of the same Agilio SmartNIC as the input port.
3. Only the input port or output ports may be a tunnel vport, not both.

For information on supported tunnel vports please see *Appendix J: Overlay Tunneling*.

Offloading of flows that output to more than one port is supported when using OVS v2.10+, as found in the Fast Datapath repository for RHEL 7. Otherwise only flows that output to at most one port may be offloaded.

Other than output and the implicit drop action, flows using the following actions may be offloaded:

1. Push and Pop VLAN
2. Masked and Unmasked Set

Flows that include a masked set of any of the following fields may be offloaded:

Layer 2	Ethernet: Type, Addresses VLAN: ID, Priority
Layer 3	IPv4: Addresses IPv4: TTL, TOS IPv6: Addresses IPv6: Hop Limit, priority
Layer 4	TCP: Ports UDP: Ports

Flows that include an unmasked set of any of the following fields may be offloaded:

Tunnel	ID IPv4: Outer Address UDP: Outer Destination Port

23 Appendix I: Quality of Service

Offload of OVS Quality of Service (QoS) rate limiting is supported when applied to VFs.

Minimum supported versions:

	Bit-Rate Limiting
Kernel	5.2
Firmware	AOTC-2.10.A.38
OVS	2.12
RHEL 7	Not Supported
RHEL 8	8.2
Ubuntu	20.04

23.1 Configuring Quality of Service (QoS) Rate Limiting with OVS

OVS has support for using policing to enforce an ingress rate limit in kilobits per second. For example, to set a rate limit of `1000` *kbps* with of burst of `100` *kbps* on `enp3s0v0`, use these commands to set the rate limit for the VF corresponding to VF representor `eth4`:

```
# ovs-vsctl set interface eth4 ingress_policing_rate=1000
# ovs-vsctl set interface eth4 ingress_policing_burst=100
```

The following command may be used to check the current rate limit configuration in OVSDB:

```
# ovs-vsctl list interface eth4 | grep ingress_policing
ingress_policing_burst: 100
ingress_policing_rate: 1000
```

The following command may be used to check the current rate limit configuration in the kernel and offload hardware:

```
# tc -s -d filter show dev eth4 ingress
eth4 ingress filter protocol
all pref 1 matchall chain 0
filter protocol all pref 1 matchall
 chain 0 handle 0x1 in_hw (rule hit 2)
   action order 1: police 0x2 rate 1Mbit burst 1600b mtu 64Kb
    action drop/continue overhead 0b linklayer unspec
```

(continues on next page)

(continued from previous page)

```
ref 1 bind 1 installed 226
sec used 0 sec Action
statistics:
Sent 260 bytes 4 pkt (dropped 0, overlimits 0 requeues 0)
Sent software 112 bytes 2 pkt
Sent hardware 148 bytes pkt backlog 0b 0p requeues0
```

24 Appendix J: Overlay Tunneling

24.1 Introduction

OVS-TC supports offloading tunnels. The supported tunnel types and the corresponding minimum versions of the various components are documented below. The OVS documentation can be referred to for more detailed information on how OVS works with tunnels, and this section will only provide a short summary of the two configurations for which offloading is supported.

24.1.1 Method 1: IP-on-the-Port

This is the simplest method, where the tunnel IP is placed on the physical port, and the port itself is not placed on the OVS bridge. The OVS bridge contains the VF representor ports, as well as a tunnel port. OVS uses Linux routing to be able to map the tunnel to the correct physical port, and uses this information to generate a datapath rule which is offloaded.

The configuration of a tunnel port will vary slightly for the different port types, refer to the specific tunnel sections below - for this section a shortened format will be use to explain the concept. The steps to configure this are as follows.

Configure the port IP address:

```
# ip addr add dev <phy0> <local_tun_ip/mask>
# ip link set dev <phy0> up
```

Configure the bridge:

```
# ovs-vsctl add-br br0
# ovs-vsctl add-port br0 vtep -- <vtep specific settings...>
# ovs-vsctl add-br br0 <vf0_repr>
```

This is all that is required to configure the underlay for successful tunneling. A simple test would be to add an IP to the VF netdev (or interface in the VM if that is used), and ping a VM/netdev on the remote machine:

```
# ip addr add dev <vf0_netdev> <local_vm_ip/mask>
# ping <remote_vm_ip>
```

24.1.2 Method 2: IP-on-the-Bridge

This is the method that is typically configured by OpenStack, and usually involves two bridges. As the name suggests the tunnel IP in this case is placed on the bridge port. A common convention is to have the two bridges called `br-ex` and `br-int`. `br-ex` will have the physical port added to it, and the IP will be placed on the `br-ex` port. `br-int` will be configured exactly the same as `br0` in *Method 1: IP-on-the-Port*.

Configure bridge `br-ex`:

```
# ovs-vsctl add-br br-ex
# ovs-vsctl add-port br-ex <phy0>

# ip addr add dev br-ex <local_tun_ip/mask>
# ip link set dev br-ex up
```

Configure bridge `br-int`:

```
# ovs-vsctl add-br br-int
# ovs-vsctl add-port br-int vtep -- <vtep specific settings...>
# ovs-vsctl add-br br-int <vf0_repr>
```

At this point the configuration is done, and can also be verified as explained in *Method 1: IP-on-the-Port*.

Note: For best behavior it is important that *action=NORMAL* is used on `br-ex`. Any more specific rules are usually applied to `br-int`.

24.2 VXLAN Tunnels

Minimum supported versions:

Kernel	4.15
Firmware	0AOTC28A.5642
OVS	2.8
RHEL 7	7.5
RHEL 8	8.0
Ubuntu	18.04 LTS

Offload of VXLAN Tunnels is supported when using UDP port 4789.

Add a VXLAN VTEP to an OVS bridge (in this case br0, assuming br0 already has an attached SR-IOV VF representor) as follows:

```
# ovs-vsctl add-port br0 vxlan0 -- set interface vxlan type=vxlan \
options:local_ip=<local_ip> options:remote_ip=<remote_ip> \
options:key=<tunnel_key>
```

The resultant flow can be seen by querying the VF representor's TC filter (with remote and local underlay IPs on subnet 10.0.0.0/24 and a tunnel key = 100):

```
# tc -s filter show ingress dev eth1
...
in_hw in_hw_count 1
    action order 1: tunnel_key  set
    src_ip 10.0.0.2
    dst_ip 10.0.0.1
    key_id 100
...
```

24.3 GENEVE Tunnels

Minimum supported versions:

	Without Options	With Options
Kernel	4.16	4.19
Firmware	AOTC-2.9.A.16	AOTC-2.9.A.31
OVS	2.8	2.11
RHEL 7	7.6	7.7
RHEL 8	8.0	8.0
Ubuntu	18.10	19.04

Offload of GENEVE Tunnels is supported when using UDP port 6801.

A GENEVE VTEP may be added to an OVS bridge in the same manner as a VXLAN VTEP:

```
# ovs-vsctl add-port br0 geneve0 -- set interface geneve type=geneve \
options:local_ip=<local_ip> options:remote_ip=<remote_ip> \
options:key=<tunnel_key>
```

The successfully offloaded flows can be queried in the VF representors' TC filter as per the example given for VXLAN.

24.4 GRE Tunnels

Minimum supported versions:

Kernel	5.3
Firmware	0AOTC28A.5642
OVS	2.11
RHEL 7	Not supported
RHEL 8	8.2
Ubuntu	19.10

A GRE VTEP may be added to an OVS bridge in the same manner as a VXLAN or GENEVE VTEP:

```
# ovs-vsctl add-port br0 gre0 -- set interface gre0 type=gre \
options:local_ip=<local_ip> options:remote_ip=<remote_ip> \
options:key=<tunnel_key>
```

The successfully offloaded flows can be queried in the VF representors' TC filter as per the example given for VXLAN.

24.5 IPv6 on the Underlay

Minimum supported versions:

Kernel	5.10
Firmware	AOTC-2.14.A.6
OVS	2.11
RHEL 7	Not supported
RHEL 8	Not released yet
Ubuntu	Not released yet

All the tunnel types mentioned above supports IPv4 since their first introduction. Support for using IPv6 has been added later as indicated in the version box above. This is valid for all the supported tunnel types mentioned so far in this section. The way to configure this is exactly the same as with IPv4, the only difference is that *<local_ip>* and *<remote_ip>* used in the example snippets are now allowed to be IPv6.

25 Appendix K: Link Aggregation (LAG)

25.1 Using Native Open vSwitch LAG

Minimum supported versions:

Kernel	4.13
Firmware	0AOTC28A.5642
OVS	2.8
RHEL 7	7.5
RHEL 8	8.0
Ubuntu	18.04 LTS

Flows resulting from the following modes could be accelerated:

OVS Bonds Modes	Active Backup Balance SLB Balance TCP

Configuring a LAG in OVS in `active-backup` or `balance-slb` modes results in flows that are offloadable.

It should be noted that by default OVS sends packets to the LOCAL port for each LAG. This results in flow rules that include actions with output to the LOCAL port. Such flows cannot be accelerated by Agilio OVS. To prevent this from occurring, and to achieve offload, packets must not be sent to the LOCAL port. This can be achieved with the command:

```
# ovs-ofctl -O Openflow13 mod-port lagbr0 lagbr0 no-forward
```

Furthermore, configuring a LAG in `balance-tcp` mode will result in flows that are offloadable unless recirculation has been disabled. This can be achieved using the following command:

```
# ovs-appctl dpif/set-dp-features lagbr0 recirc false
```

It should be noted that turning off recirculation leads to exact match datapath entries (matching on L2, L3 and L4) being installed. This can be seen when running the following command:

```
# ovs-appctl dpctl/dump-flows
```

Expected output from the above:

```
in_port(10),eth(src=12:23:34:45:56:67,dst=67:56:45:34:23:12),eth_type(0x0800),
↪ipv4(src=10.10.10.10,dst=10.10.10.20,proto=6,frag=no),tcp(src=1000,dst=2000),␣
↪packets:0, bytes:0, used:never, actions:6,7
```

This exact matching behavior leads to flow explosion, i.e. OVS will install an entry for every unique (L2, L3 or L4) packet. This in turn could lead to performance degradation, especially when using many flows (100K and more).

Finally, OVS LAG is based on the NORMAL rule; links will not be aggregated when the LAG bridge does not contain a NORMAL rule. Should match/actions be required, an additional bridge (named `br0` in this example) is required on which the match/actions are performed, allowing the LAG bridge to only have the NORMAL rule. This additional bridge can be connected to the LAG bridge using a patch port.

25.2 Configuring Linux Bond LAGs

Minimum supported versions:

Kernel	4.18
Firmware	AOTC-2.9.A.16
OVS	2.10
RHEL 7	7.7
RHEL 8	8.0
Ubuntu	18.10

It is possible to configure standard Linux bonds and add them to an OVS bridge for offloading. The process to create and use these LAGs are shown next.

First create a bond LAG device:

```
# ip link add lag0 type bond
```

Add the physical port representor ports to the LAG:

```
# ip link set dev ens1np0 master lag0
# ip link set dev ens1np1 master lag0
```

If they need to be removed from the LAG:

```
# ip link set dev ens1np0 nomaster
# ip link set dev ens1np1 nomaster
```

Information about a Linux LAG can be obtained by:

```
# cat /proc/net/bonding/lag0
```

Example of the output from the above command:

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: load balancing (round-robin)
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0

Slave Interface: ens1np0
MII Status: up
Speed: 40000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:15:4d:13:50:32
Slave queue ID: 0

Slave Interface: ens1np1
MII Status: up
Speed: 40000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 00:15:4d:13:50:33
Slave queue ID: 0
```

Not all bond LAG modes are supported for offloading. The currently supported modes are active-backup and balance-xor. See below for more info configuring each mode.

Note: All lower-devices need to be removed from a bond LAG device before the mode can be changed.

25.2.1 Active-backup

The `active-backup` mode will send traffic on only one of the ports that are aggregated in the LAG. This mode is configured by executing:

```
# ip link set dev lag0 down
# ip link set dev ens1np0 nomaster lag0
# ip link set dev ens1np1 nomaster lag0
# ip link set dev lag0 type bond mode active-backup
# ip link set dev lag0 type bond miimon 100
# ip link set dev ens1np0 master lag0
# ip link set dev ens1np1 master lag0
# ip link set dev lag0 up
```

The `miimon` setting sets the interval on which the link state should be monitored in milliseconds. If a port down state is detected the LAG will reconfigure itself to send the traffic out on one of the other ports in the LAG.

25.2.2 Balance-xor

The `balance-xor` mode balances traffic across the aggregated ports using a hash method. To enable offloading the `xmit_hash_policy` value must be set to either `layer3+4` or `encap3+4`. Other hashing methods will not be offloaded. Configuration is as follows:

```
# ip link set dev lag0 down
# ip link set dev ens1np0 nomaster
# ip link set dev ens1np1 nomaster
# ip link set dev lag0 type bond mode balance-xor
# ip link set dev lag0 type bond miimon 100
```

To use `layer3+4` as hash:

```
# ip link set dev lag0 type bond xmit_hash_policy layer3+4
```

To use `encap3+4` as hash:

```
# ip link set dev lag0 type bond xmit_hash_policy encap3+4
```

Add back the lower-devices and up the LAG:

```
# ip link set dev ens1np0 master lag0
# ip link set dev ens1np1 master lag0
# ip link set dev lag0 up
```

For more detailed information on the difference between the modes and the hash methods it is recommended to read the Linux kernel documentation on the subject.

25.3 Configuring Linux Teaming

Another method of setting up link aggregated ports is to use Linux teaming. Teaming is controlled using the `teamd` and `teamdctl` utilities, as will be demonstrated below.

Creating a new team device for active-backup mode:

```
# teamd -t lag0 -d -c '{"runner": {"name": "activebackup"}}'
```

Creating a new team device for load balancing mode. The hashing method for teaming is not as well defined so for offloading to the NFP this will hash on L3 and L4:

```
# teamd -t lag0 -d -c '{"runner": {"name": "lacp"}}'
```

Ports are added using `teamdctl`:

```
# teamdctl lag0 port add ens6np0
# teamdctl lag0 port add ens6np1
```

The port config can be dumped using:

```
# teamdctl lag0 config dump
```

Example output:

```
{
    "device": "lag0",
    "ports": {
        "ens6np0": {
            "link_watch": {
                "name": "ethtool"
            }
        },
        "ens6np1": {
            "link_watch": {
                "name": "ethtool"
            }
        }
    },
    "runner": {
        "name": "lacp",
        "tx_hash": [
            "eth",
            "ipv4",
            "ipv6"
        ]
    }
}
```

For more usage instructions using teaming take a look at the man pages for `teamd` and `teamdctl`.

25.4 Using Linux LAG with Open vSwitch

Once the LAG is configured as shown in section *Configuring Linux Bond LAGs*, it is possible to use it with Open vSwitch by adding the LAG port to the bridge as with any other type of port. See the following example which adds a bridge, configures the LAG port as well as a VF representor port and then adds two simple flow rules that forwards all traffic between the VF and the LAG:

```
# ovs-vsctl add-br br0
# ovs-vsctl add-port br0 lag0
# ovs-vsctl add-port br0 <vf0_repr>
# ovs-ofctl add-flow br0 in_port=lag0,actions=output:<vf0_repr>
# ovs-ofctl add-flow br0 in_port=<vf0_repr>,actions=output:lag0
```

Teams are used with Open vSwitch in exactly the same way as Linux bond LAGs.

25.5 Using Linux LAG with Tunnels

Minimum supported versions:

Kernel	5.2
Firmware	AOTC-2.10.A.23
OVS	2.11
RHEL 7	7.7
RHEL 8	8.0
Ubuntu	19.10

It is possible to configure tunnels to work in conjunction with Linux LAG ports as of kernel 5.2. The simplest way to configure this is to make use of two OVS bridges. Add the tunnel port to the first bridge, the LAG port to the second bridge and add the tunnel endpoint IP to the second bridge. Refer to *Method 2: IP-on-the-Bridge* to see how this is configured.

The only difference is that instead of placing `phy0` on `br-ex` the LAG port is placed on the bridge:

```
# ovs-vsctl add-br br-ex
# ovs-vsctl add-port br-ex lag0
```

The rest of the config stays the same.

26 Appendix L: QinQ

Minimum supported versions:

	QinQ offload
Kernel	5.10
Firmware	AOTC-2.14.A.6
OVS	2.11
RHEL 7	Not Supported
RHEL 8	Not released yet
Ubuntu	Not released yet

26.1 Configuring QinQ in OVS

OVS has support to configure QinQ, previously known as 802.1ad. Support to offload this has been added from the versions above and later. There are two ways to configure this. The first is to use OVS port types together with the `NORMAL` rule. Enable the feature:

```
# ovs-vsctl set Open_vSwitch . other_config:vlan-limit=2
```

Next, configure the port with ovs-vsctl to add a service tag (outer VLAN) for specific customer tags (inner VLAN):

```
# ovs-vsctl set port <phy0_repr> vlan_mode=dot1q-tunnel tag=2000 cvlans=100
```

As mentioned above, this only works when using `actions=NORMAL`. An alternative method is to use OpenFlow rules to push and pop VLAN tags, similarly to how it would be done with just a single VLAN.

Note: It is still required to set vlan-limit=2, even if using OpenFlow rules directly.

Adding a VLAN tag can be achieved with the following command:

```
# ovs-ofctl add-flow br0 \
  in_port=<repr 1> actions=push_vlan:0x88a8,mod_vlan_vid=2000,output:<repr 2>
```

The above will push a tag with type 0x88a8, and vlan_id=2000 onto a packet. It is also possible to push both an inner and outer VLAN tag in the same action:

```
# ovs-ofctl add-flow br0 \
  in_port=<repr 1>,actions=push_vlan:0x8100,mod_vlan_vid=200, \
  push_vlan:0x88a8,mod_vlan_vid=2000,output:<repr 2>
```

This will push an inner tag of type 0x8100 and vlan_id 200, as well as an outer tag with type 0x88a8 and vlan_id 2000. This is a slightly unusual use case, normally the traffic will already have an inner tag, and just the outer tag needs to be pushed.

Removing a tag is quite easy:

```
# ovs-ofctl add-flow br0 in_port=<repr 2>,actions=pop_vlan,output:<repr 1>
```

There is no way to specify which tag needs to be stripped, so the pop_vlan action will always remove the most outer VLAN. Once again it is possible to remove both tags with a single rule, just chain the `pop_vlan` actions:

```
# ovs-ofctl add-flow br0 in_port=<repr 2>,actions=pop_vlan,pop_vlan,output:<repr 1>
```

Note: Only a maximum of two tags is supported for offloading. Another limitation is that while a single VLAN tag on the outside of a tunnel header is supported for offloading, this is not supported with multiple tags.

27 Appendix M: UEFI Secure Boot with Out-of-Tree NFP Driver

UEFI secure boot ensures that only kernel modules signed with trusted keys can be loaded.

When the NFP driver module is loaded without a signature or with an invalid signature, the errors below may appear:

```
# dmesg
Lockdown: modprobe: unsigned module loading is restricted; see man kernel_lockdown.7
# modprobe nfp
modprobe: ERROR: could not insert 'nfp': Required key not available
```

If the errors above occur when loading the NFP driver, please check the signature of the NFP driver module:

```
# modinfo nfp | grep sig
```

Conditions may as follow:

- If the NFP driver is signed with a DKMS module signing key, as below, please refer to *NFP Driver Module is Signed with a DKMS Module Signing key*:

  ```
  # modinfo nfp | grep sig
  signer:         DKMS module signing key
  ```

- If nothing is printed with command above, then the NFP driver module is not signed, please refer to *NFP Driver Module is Not Signed or Signed with Unknown Keys*.
- If the NFP driver is signed with an unknown key, please refer to *NFP Driver Module is Not Signed or Signed with Unknown Keys*.

27.1 NFP Driver Module is Signed with a DKMS Module Signing key

DKMS supports module signing from version 2.8.1 for Ubuntu/Debian and version 3.0.4 for other OS. More details can be found in the DKMS repository (https://github.com/dell/dkms/blob/master/README.md#module-signing).

27.1.1 RHEL and CentOS

The public key is placed in `/var/lib/dkms` by default. Enroll the public key to the MOK list. You may need to set a passphrase for the enrollment:

```
# mokutil --import /var/lib/dkms/mok.pub
```

Reboot the system and enter the enrollment and confirm the passphrase. The system will then boot normally and the NFP driver can be loaded.

27.1.2 Ubuntu

The public key is placed in `/var/lib/shim-signed/mok`. The enrollment is executed by automatically during driver installation via software package. Following the guidance to set the passphrase for the enrollment and reboot is required to finish enrollment.

27.2 NFP Driver Module is Not Signed or Signed with Unknown Keys

When the NFP module is not signed or signed with keys which can't be found. A pair of new keys for module signing may be generated.

1. Key generation

   The Machine Owner Key (MOK) can be pre-generated and distributed or generated on the target machine using OpenSSL:

   ```
   # openssl req -x509 -nodes -days 36500 -subj "/CN=Secure Boot DKMS Signing
   ↪"
   -newkey rsa:2048 -keyout /root/MOK.priv -outform DER -out /root/MOK.der
   ```

   Note:

   - The key generated here is not encrypted with parameter `-nodes`.
   - If additional security is required, please refer to the related document in Ubuntu (https://ubuntu.com/blog/how-to-sign-things-for-secure-boot).

2. Key enrollment

   MOK enrollment process for the generated keys:

   ```
   # mokutil --import /root/MOK.der
   ```

   Reboot the system and finish the enrollment.

3. Manual DKMS module signing

The NFP module can be signed with the enrolled keys.

Check the name of module:

```
# modinfo nfp | grep filename
```

If the module name ends with `.xz`, f.e. `nfp.ko.xz`, then you may need to decompress it:

```
# NFP_DRV_MODULE=$(modinfo nfp | grep filename | awk -F ' ' '{print $2}')
# xz -d ${NFP_DRV_MODULE}
# depmod -a
```

For RHEL and CentOS:

```
# NFP_DRV_MODULE=$(modinfo nfp | grep filename | awk -F ' ' '{print $2}')
# /lib/modules/$(uname -r)/build/scripts/sign-file sha256 /root/MOK.priv
/root/MOK.der ${NFP_DRV_MODULE}
```

For Ubuntu:

```
# NFP_DRV_MODULE=$(modinfo nfp | grep filename | awk -F ' ' '{print $2}')
# kmodsign sha256 /root/MOK.priv /root/MOK.der ${NFP_DRV_MODULE}
```

Or:

```
# NFP_DRV_MODULE=$(modinfo nfp | grep filename | awk -F ' ' '{print $2}')
# /usr/src/linux-headers-$(uname -r)/scripts/sign-file sha256 /root/MOK.
↪priv
/root/MOK.der ${NFP_DRV_MODULE}
```