Technical Support

This site is for customers who have purchased Netronome's Agilio SmartNICs and software products in 2015 and beyond. The **Netronome Legacy Support Site (https://support.netronome.com/)** ensures seamless support for existing customers who have deployed Netronome Flow Processor solutions prior to 2015.

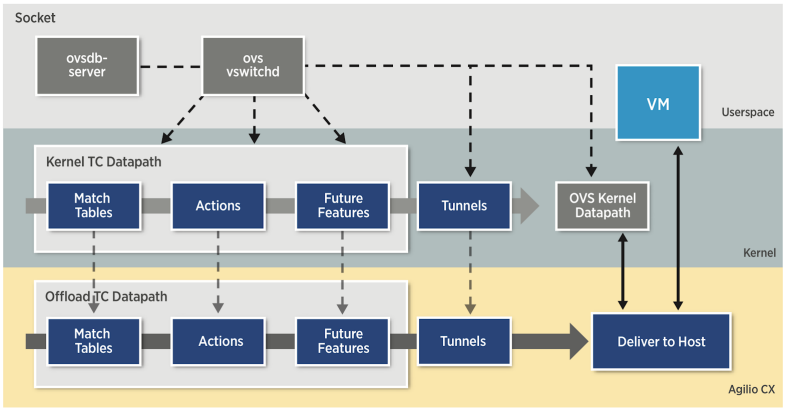
Agilio Open vSwitch TC User Guide

Modified on: Mon, Apr 26, 2021 at 1:56 PM

Agilio Open vSwitch
Contents:

Agilio OVS with TC Offload User's Guide¶

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**.*

Hardware Installation¶

*This user guide focuses on x86 deployments of Open vSwitch hardware acceleration in supported versions of Ubuntu 18.04, Red Hat Enterprise Linux (RHEL) 7.5+, and CentOS 7.5+. As detailed in **Validating the Driver**, Netronome'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

Identification¶

The serial number is printed beside a bar code on the outside of the card and is of the form `SMAAMDAXXXX-XXXXXXXXXXX`. The `AMDAXXXX` section denotes the assembly ID. In a running system, the assembly ID and serial number of a PCI device may be determined by using the `ethtool` debug interface, if the netdev associated with the SmartNIC is known, by passing it as the first argument to the following script:

```
#!/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}
```

*Consult **SmartNIC Netdev Interfaces** for methods identifying the netdev.*
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.

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. Further details and support about the hardware installation process can be reviewed in the Hardware User Manual available from Netronome's support site.

Validation¶

Use the following command to validate that the SmartNIC is being correctly detected by the host server and to identify its PCI address:

```
# lspci -Dnnd 19ee:4000; lspci -Dnnd 19ee:6000
0000:02:00.0 Ethernet controller [0200]: Netronome Systems, Inc. Device    [19ee:4000]
```

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.

Validating the Driver¶

The Netronome 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 is required please contact Netronome support: **Contact Us**.*

Confirm Upstreamed NFP Driver¶

To confirm that your current operating system supplies the upstreamed nfp module, you can use the `modinfo` command:

```
# 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 distributions and kernel version to a version that includes the upstreamed drivers.*

Confirm that the NFP Driver is Loaded¶

Use `lsmod` to list the loaded driver modules and look 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
```

Validating the Firmware¶

Netronome 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
-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 H: 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 enp4s0np0
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
```

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.

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 offloaded firmware is located in the `netronome/flower` directory in `lib/firmware`.
*Review **SmartNIC Netdev Interfaces** to identify the SmartNIC's netdev. The script in **Identification** details how to identify the SmartNIC's assembly.*
The following script extract illustrates how to create and persist this symbolic link:

```
#!/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 Netronome 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
```

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

As an example:

- *The script has been assembled into* `./agilio-tc-fw-select.sh`
- *A netdev associated with the SmartNIC is* `p5p1`
- *The user wishes to auto-detect the Assembly ID*

```
# ./agilio-tc-fw-select.sh p5p1 scan
# rmmod nfp
# modprobe nfp
```

*If the out-of-tree firmware repository has been installed (as described in **Appendix H: Upgrading TC Firmware**) and the user wishes to select that instead:*

```
# ./agilio-tc-fw-select.sh p5p1 scan flower-next
# rmmod nfp
# modprobe nfp
```

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 ens3np0
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
```

SmartNIC Netdev Interfaces¶

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 to allow configuration, for example of link state, of the port, to access statistics of the port and to 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, and thus 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.

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. These PCI devices may be determined using the `lspci` tool to list devices with Netronome vendor:device tuples (`19ee:4000` and `19ee:6000`). The netdevs associated with these devices may be determined by examining sysfs:

```
1 #!/bin/bash
2 BDFS=$({ lspci -Dmmd 19ee:4000; lspci -Dmmd 19ee:6000; } | cut -f 1 -d " ")
3 for i in $BDFS; do ls /sys/bus/pci/drivers/nfp/$i/net/; done
```

An example output of this would be:

```
enp4s0np0  enp4s0np1  p6p1
```

Where `enp4s0np0` and `enp4s0np1` are the physical port representors and `p6p1` 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 `p6p1`) will report an error.

```
# cat /sys/class/net/p6p1/phys_port_name
cat /sys/class/net/p6p1/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 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]`. To 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
```

Support for `biosdevname`¶

Netronome 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.

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 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
```

Settings¶

RHEL 7.5+ and CentOS 7.5+¶

NetworkManager may be configured to bring up a device without addresses as follows. NetworkManager may not present on some installs (check with `systemctl status NetworkManager.service`), it can be installed using `yum`:

```
# yum install NetworkManager
```

In this example, the device is `p5p1` (replace this to match the PF netdev in question). First add the connection type to NetworkManager, then change IP configurations as follows:

```
# nmcli c add type ethernet ifname p5p1 con-name ethernet-p5p1
Connection 'ethernet-p5p1' (0e3e4e76-f592-4814-963b-e3fbecf00504) successfully added.
# nmcli c modify ethernet-p5p1 ipv4.method disabled
# nmcli c modify ethernet-p5p1 ipv6.method ignore
# nmcli c modify ethernet-p5p1 ethernet.mtu 9240
# nmcli c modify ethernet-p5p1 connection.autoconnect yes
```

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.
NetworkManager 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.

```
# nmcli c up ethernet-p5p1
```

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 **Configuring SR-IOV***

Ubuntu 18.04¶

*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 `p5p1` 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 p5p1
ip link set up dev p5p1
EOF
chmod u+x /usr/lib/networkd-dispatcher/routable.d/50-ifup-noaddr
```

In order to ensure the hook above is run, regardless if 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)
       Tasks: 2 (limit: 7372)
           CGroup: /system.slice/networkd-dispatcher.service
                   └─41757 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
```

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.

Verification¶

Verify link state and MTU of the PF netdev. For example the netdev `p5p1` (unlike the physical port representors `enp4s0np0` or `enp4s0np1`) outputs:

```
# ip addr show p5p1
14: p5p1: <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
```

Install Open vSwitch¶

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	*Version*
RHEL 7.5	`2.9.0-19.el7fdp`
CentOS 7.5+	`2.9.0-3.el7`
RHEL 7.6	`2.9.0-55.el7fdp`
RHEL 7.7	`2.11.0-14.el7fdp`
RHEL 8.0	`2.11.0-9.el8fdp`
Ubuntu 18.04 LTS	`2.9.0-0ubuntu1`

RHEL 7.5+¶

*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 openvswitch
```

CentOS 7.5+¶

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 yum-utils
# yum install centos-release-openstack-rocky
```

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 --enablerepo centos-openstack-rocky openvswitch
```

At the time of writing this will install `openvswitch-2.10.1-3`.

Ubuntu 18.04 LTS¶

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

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

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**.

Installation from Source¶

Installing Open vSwitch from source is only recommended for developers, the official **Open vSwitch Git Repository** (https://github.com/openvswitch/ovs) contains instructions on how to build the packages from the source for each supported operating system.

Using the Linux Driver¶

Configuring SR-IOV¶

To configure 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 an 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 Netronome 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.
In order to persist the virtual functions on the system, it is suggested that the system networking scripts be updated to manage them. The following snippet illustrates how to implement such a configuration with NetworkManager for the physical function `ens3`:

```
1  cat >/etc/NetworkManager/conf.d/nfp.conf << EOF
2  [keyfile]
3  unmanaged-devices=driver:nfp,driver:nfp_netvf,except:interface-name=ens3
4  [device]
5  match-device=interface-name:ens3
6  sriov-num-vfs=4
7  EOF
8  systemctl restart NetworkManager
```

This will setup NetworkManager to create 4 VF interfaces connected to the PF on `ens3`.
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.
In Ubuntu systems, networkd-dispatcher can be used in place of NetworkManager, as demonstrated below:

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

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.

Configuring interface link-speed¶

The following steps explains how to change between 10G mode and 25G mode on Agilio CX 2x25GbE cards. The changing of port speed must be done in order, p0 must be set to 10G before p1 may be set to 10G.
Down respective interface(s):

```
# ip link set dev enp4s0np0 down
```

Set interface link speed to 10G:

```
# ethtool -s enp4s0np0 speed 10000
```

Alternatively, set interface link speed to 25G:

```
# ethtool -s enp4s0np0 speed 25000
```

Reload driver for changes to take effect:

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

Note
The settings above only apply to Agilio CX 25G SmartNICs and older drivers/firmware changes may require a system reboot for changes to take effect

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.
Set interface `ens3np0`'s MTU to 9000 bytes:

```
# ip link set dev ens3np0 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.

Configuring FEC modes¶

Agilio CX 2x25GbE SmartNICs support FEC mode configuration, e.g. Auto, Firecode BaseR, 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 ens3np0
Settings for ens3np0:
    Supported ports: [ FIBRE ]
    Supported link modes:   Not reported
    Supported pause frame use: No
    Supports auto-negotiation: No
    Supported FEC modes: None BaseR RS
    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 Agilio CX 2x25GbE SmartNIC used for the example above only supports Firecode BaseR 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 Netronome upstream kernel driver provides `ethtool` FEC support from kernel 4.15. Furthermore, the SmartNIC NVRAM version must be at least `020025.020025.02006e` 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'
[2387.682046] nfp 0000:82:00.0: BSP: 020025.020025.020072
```

*This example lists a version of `020025.020025.020072` which is sufficient to support `ethtool` FEC mode configuration. To update your SmartNIC NVRAM flash, refer to **Appendix E: Updating NFP Flash** or contact **Netronome 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 enp130s0np0
Settings for enp130s0np0:
...
Supported FEC modes: None
```

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

```
# ethtool --show-fec enp130s0np0
FEC parameters for enp130s0np0:
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 enp130s0np0 down
# ethtool -s enp130s0np0 autoneg off
# ip link set enp130s0np0 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 BaseR:

```
# ethtool --set-fec enp130s0np0 encoding baser
```

Verify the newly selected mode:

```
# ethtool --show-fec enp130s0np0
FEC parameters for enp130s0np0:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: BaseR
```

To modify the FEC mode to Reed Solomon:

```
# ethtool --set-fec enp130s0np0 encoding rs
```

Verify the newly selected mode:

```
# ethtool --show-fec enp130s0np0
FEC parameters for enp130s0np0:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: RS
```

To modify the FEC mode to Off:

```
# ethtool --set-fec enp130s0np0 encoding off
```

Verify the newly selected mode:

```
# ethtool --show-fec enp130s0np0
FEC parameters for enp130s0np0:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: Off
```

Revert back to the default Auto setting:

```
# ethtool --set-fec enp130s0np0 encoding auto
```

Verify the setting again:

```
# ethtool --show-fec enp130s0np0
FEC parameters for enp130s0np0:
Configured FEC encodings: Auto Off BaseR RS
Active FEC encoding: Auto
```

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

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 Agilio CX 40GbE and CX 2x40GbE SmartNICs.
Determine the card's PCI address:

```
# lspci -Dkd 19ee:4000
0000:04:00.0 Ethernet controller: Netronome Systems, Inc. Device 4000
    Subsystem: Netronome Systems, Inc. Device 4001
    Kernel driver in use: nfp
    Kernel modules: nfp
```

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 change 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 ag
[ 6270.553902] nfp 0000:04:00.0: nfp: Port #4 config changed, unregistering. Driver reload required before port will be operational ag
```

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 Agilio 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.

Confirming Connectivity¶

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 by:

```
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
```

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
```

Basic Firmware Features¶

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

View Interface Parameters¶

The `-k` flag can be used to view current interface configurations. For example, using a Agilio CX 1x40GbE NIC with a physical port representor `enp4s0np0`:

```
# ethtool -k enp4s0np0
Features for enp4s0np0:
rx-checksumming: off [fixed]
tx-checksumming: off
tx-checksum-ipv4: off [fixed]
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]
```

Setting 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. In this section,* `ens3` *will be used as an example of a physical function netdev.*

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 ens3 rx on
```

To disable receive checksum offload:

```
# ethtool -K ens3 rx off
```

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 ens3 tx on
```

To disable transmit checksum offload:

```
# ethtool -K ens3 tx off
```

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 ens3 sg on
Actual changes:
scatter-gather: on
        tx-scatter-gather: on
generic-segmentation-offload: on

# ethtool -K ens3 sg off
Actual changes:
scatter-gather: on
        tx-scatter-gather: on
generic-segmentation-offload: on
```

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 ens3 tso on
```

To disable TCP segmentation offload:

```
# ethtool -K ens3 tso off
```

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 ens3 gso on
```

To disable generic segmentation offload:

```
# ethtool -K ens3 gso off
```

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 ens3 gro on
```

To disable generic receive offload:

```
# ethtool -K ens3 gro off
```

Note
Take note that scripts that use `ethtool -i ${INTERFACE}` to get bus-info will not work on representors as this information is not populated for representor devices.

Using Open vSwitch¶

Running Open vSwitch¶

RHEL 7.5+ and CentOS 7.5+¶

Start Open vSwitch:

```
# systemctl start openvswitch
```

Check status of Open vSwitch:

```
# systemctl status openvswitch
● openvswitch.service - Open vSwitch
   Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; enabled; vendor preset: disabled)
      Active: active (exited) since Mon 2018-05-07 11:18:16 SAST; 2min 13s ago
    Process: 130744 ExecStop=/bin/true (code=exited, status=0/SUCCESS)
    Process: 131101 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 131101 (code=exited, status=0/SUCCESS)

  May 07 11:18:16 r730-dev-51 systemd[1]: Starting Open vSwitch...
  May 07 11:18:16 r730-dev-51 systemd[1]: Started Open vSwitch.
```

The openvswitch service controls the ovsdb-server and ovs-vswitchd services. Check their status too:

```
# 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 Mon 2018-05-07 11:18:16 SAST; 5min ago
 Process: 130869 ExecStop=/usr/share/openvswitch/scripts/ovs-ctl --no-ovs-vswitchd
   stop (code=exited, status=0/SUCCESS)
 Process: 130898 ExecStart=/usr/share/openvswitch/scripts/ovs-ctl --no-ovs-vswitchd
   --no-monitor --system-id=random --ovs-user=#{OVS_USER_ID} start $OPTIONS
 (code=exited, status=0/SUCCESS)
   Process: 130895 ExecStartPre=/usr/bin/chown #{OVS_USER_ID} /var/run/openvswitch
 (code=exited, status=0/SUCCESS)
Main PID: 130939 (ovsdb-server)
   Tasks: 1
  CGroup: /system.slice/ovsdb-server.service
            `-130939 ovsdb-server /etc/openvswitch/conf.db -vconsole:emer
            -vsyslog:err -vfile:info --remote=punix:/var/run/openvswitch/db.sock...


            May 07 11:18:16 r730-dev-51 systemd[1]: Starting Open vSwitch Database Unit...
            May 07 11:18:16 r730-dev-51 runuser[130932]: pam_unix(runuser:session): sess...)
            May 07 11:18:16 r730-dev-51 runuser[130932]: pam_unix(runuser:session): sess...h
            May 07 11:18:16 r730-dev-51 runuser[130936]: pam_unix(runuser:session): sess...)
            May 07 11:18:16 r730-dev-51 runuser[130936]: pam_unix(runuser:session): sess...h
            May 07 11:18:16 r730-dev-51 ovs-ctl[130898]: Starting ovsdb-server [  OK  ]
            May 07 11:18:16 r730-dev-51 ovs-vsctl[130940]: ovs|00001|vsctl|INFO|Called a...0
            May 07 11:18:16 r730-dev-51 ovs-vsctl[130946]: ovs|00001|vsctl|INFO|Called as...
            May 07 11:18:16 r730-dev-51 ovs-ctl[130898]: Configuring Open vSwitch system...]
            May 07 11:18:16 r730-dev-51 systemd[1]: Started Open vSwitch Database Unit.
            Hint: Some lines were ellipsized, use -l to show in full.
```

```
# 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 Mon 2018-05-07 11:18:16 SAST; 6min ago
   Process: 130747 ExecStop=/usr/share/openvswitch/scripts/ovs-ctl --no-ovsdb-server stop
       (code=exited, status=0/SUCCESS)
   Process: 130955 ExecStart=/usr/share/openvswitch/scripts/ovs-ctl --no-ovsdb-server
       --no-monitor --system-id=random --ovs-user=#{OVS_USER_ID} start $OPTIONS
           (code=exited, status=0/SUCCESS)
   Process: 130952 ExecStartPre=/usr/bin/chmod 0775 /dev/hugepages
       (code=exited, status=0/SUCCESS)
   Process: 130949 ExecStartPre=/usr/bin/chown :hugetlbfs /dev/hugepages
       (code=exited, status=0/SUCCESS)
 Main PID: 130987 (ovs-vswitchd)
     Tasks: 50
   CGroup: /system.slice/ovs-vswitchd.service
             `-130987 ovs-vswitchd unix:/var/run/openvswitch/db.sock -vconsole:emer
             -vsyslog:err -vfile:info --mlockall --user openvswitch:huge...


             May 07 11:18:16 r730-dev-51 systemd[1]: Starting Open vSwitch Forwarding Unit...
             May 07 11:18:16 r730-dev-51 ovs-ctl[130955]: Starting ovs-vswitchd [  OK  ]
             May 07 11:18:16 r730-dev-51 ovs-ctl[130955]: Enabling remote OVSDB managers ...]
             May 07 11:18:16 r730-dev-51 systemd[1]: Started Open vSwitch Forwarding Unit.
             Hint: Some lines were ellipsized, use -l to show in full
```

Enable Open vSwitch so that it will run on reboot:

```
# systemctl enable openvswitch
```

Ubuntu 18.04 LTS¶

Start Open vSwitch:

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

Check status of Open vSwitch:

```
# 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. Check their status too:

```
# 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
```

Enable Open vSwitch so that it will run on reboot:

```
# 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
```

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 **Setting 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`. Mote that the PF interface won't show the `hw-tc-offload` flag being set by default. For example:*

```
# ethtool -k ens3 | grep hw-tc-offload
hw-tc-offload: on
```

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

```
# ethtool -K ens3np0 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`.
Configure Open vSwitch hardware offload:

```
# 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 7.5+ and CentOS 7.5+ this is performed by the command:

```
# systemctl restart openvswitch
```

In Ubuntu 18.04, the following command is used instead:

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

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 enp4s0np0
```

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 "enp4s0np0"
            Interface "enp4s0np0"
    ovs_version: "2.9.0"

# 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-dpctl`. The port numbers used for `in_port` and the `(output)` actions correspond to those listed by `ovs-dpctl` show as shown above.
View offloaded datapath flows:

```
# ovs-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
...
```

View non-offloaded datapath flows:

```
# ovs-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:neve
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:13
...
```

View both offloaded and non-offloaded datapath flows:

```
# ovs-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:neve
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:13
...
```

Note
Note that `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 enp4s0np0
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
  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
```

Appendix A: Netronome Repositories¶

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

Importing GPG-key¶

Download and Import GPG-key to your local machine:
For RHEL/Centos 7.5+, download the public key:

```
# wget https://rpm.netronome.com/gpg/NetronomePublic.key
```

Import the public key:

```
# rpm --import NetronomePublic.key
```

For Ubuntu 18.04 LTS, download the public key:

```
# wget https://deb.netronome.com/gpg/NetronomePublic.key
```

Import the public key:

```
# apt-key add NetronomePublic.key
```

Configuring Repositories¶

For RHEL 7.5+ and CentOS 7.5+:

```
cat << 'EOF' > /etc/yum.repos.d/netronome.repo
[netronome]
name=netronome
baseurl=https://rpm.netronome.com/repos/centos/
gpgcheck=0
enabled=1
EOF
yum makecache
```

For Ubuntu 18.04 LTS:

```
mkdir -p /etc/apt/sources.list.d/
echo "deb [trusted=yes] https://deb.netronome.com/apt stable main" > \
    /etc/apt/sources.list.d/netronome.list
apt-get update
```

Appendix B: Red Hat Repositories¶

TC offload is only available in Open vSwitch version 2.8, with additional offloads enabled in 2.9. 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.
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 7.5+:

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

RHEL 8.0+:

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

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

The nfp driver can be installed via the Netronome 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.

Install Driver via Netronome Repository¶

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

Ubuntu 18.04 LTS¶

```
# apt-cache search nfp-driver
agilio-nfp-driver-dkms - agilio-nfp-driver driver in DKMS format.

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

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 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** (https://github.com/dell/dkms/issues/14)). This is worked around by the package install step by adding `--force` to the `dkms` install, 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:

```
1 mkdir -p /etc/depmod.d
2 echo "override nfp * extra" > /etc/depmod.d/netronome.conf
3 depmod -a
4 modprobe -r nfp; modprobe nfp
5 update-initramfs -u
```

Building from Source¶

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

Ubuntu 18.04 Dependencies¶

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

Clone, Build and Install¶

```
1 git clone https://github.com/Netronome/nfp-drv-kmods.git
2 cd nfp-drv-kmods
3 make
4 make install
5 depmod -a
```

Appendix D: Working with Board Support Package¶

The NFP BSP provides infrastructure software and a development environment for managing NFP based platforms.

Install Software from Netronome Repository¶

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

RHEL 7.5+ and CentOS 7.5+¶

```
# yum list available | grep nfp-bsp
nfp-bsp-6000-b0.x86_64                  2018.10.31.1229-1          netronome

# yum install nfp-bsp-6000-b0 --nogpgcheck
# reboot
```

Ubuntu 18.04 LTS¶

```
# apt-cache search nfp-bsp
nfp-bsp-6000-b0 - Netronome NFP BSP

# apt-get install nfp-bsp-6000-b0
```

Install Software From deb/rpm Package¶

Obtain Software¶

*The latest BSP packages can be obtained at the downloads area of the Netronome Support site (**https://help.netronome.com** (https://help.netronome.com)).*

Install the prerequisite dependencies¶

RHEL 7.5+ and CentOS 7.5+ Dependencies¶

No dependency installation required

Ubuntu 18.04 LTS Dependencies¶

To install the BSP package dependencies on Ubuntu 18.04, run:

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

NFP BSP Package¶

Install the NFP BSP package provided by Netronome Support.

RHEL 7.5+ and CentOS 7.5+ Install¶

```
# yum install -y nfp-bsp-6000-*.rpm --nogpgcheck
```

Ubuntu 18.04 LTS Install¶

```
# dpkg -i nfp-bsp-6000-*.deb
```

Using BSP tools¶

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 Netronome 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, a number of options are available; the distribution specific documentation will detail that process more thoroughly. Care must be taken that the settings are also applied to any initramfs images generated.

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 use the following commands. Note, a reboot is still required for changes to take effect.*

Agilio 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.*

Agilio CX 1x40GbE - AMDA0081¶

Set port 0 to 40G mode:

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

Set port 0 to 4x10G fanout mode:

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

Agilio 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
```

Appendix E: Updating NFP Flash¶

The NVRAM flash software on the SmartNIC can be updated in one of two ways, either via `ethtool` or via the BSP userspace tools. In both cases, the BSP package needs to be installed to gain access to the intended flash image. After the flash has been updated, the system needs to be rebooted to take effect.
Note
The `ethtool` interface is only available for hosts running kernel 4.16 or higher when using the in-tree driver. Please use the out of tree driver to enable `ethtool` flashing on older kernels.
Note
*Updating the flash via `ethtool` is only supported if the existing flash version is greater than `0028.0028.007c`. Installed NVRAM flash version can be checked with the command `dmesg | grep BSP`. Cards running older versions of the NVRAM flash must be updated using the method in **Update via BSP Userspace Tools***
*Refer to **Appendix D: Working with Board Support Package** to acquire the BSP tool package.*

Update via Ethtool¶

To update the flash using `ethtool`, the reflashing utilities used in the Netronome directory in the system must first be relocated so that `ethtool` has access to them:

```
# cp /opt/netronome/flash/flash-nic.bin /lib/firmware
# cp /opt/netronome/flash/flash-one.bin /lib/firmware
```

Thereafter, `ethtool` can be used to reflash the software loaded onto the SmartNIC devices identified by either their PF netdev or their physical port representor:

```
# ethtool -f ens3 flash-nic.bin
# ethtool -f ens3 flash-one.bin
# reboot
```

Update via BSP Userspace Tools¶

Obtain Out of Tree NFP Driver¶

*To update the flash using the BSP userspace tools, use the following steps. Refer to **Appendix C: Installing the Out-of-Tree NFP Driver** on installing the out of tree NFP driver and to load the driver with CPP access.*

Flash the Card¶

The following commands may be executed for each card installed in the system using the PCI address of the particular card. In this section, the card's PCI address is assumed to be `0000:04:00.0`. First reload the NFP drivers with CPP access enabled:

```
# rmmod nfp
# modprobe nfp nfp_pf_netdev=0 nfp_dev_cpp=1
```

Then use the included Netronome flashing tools to reflash the card:

```
# /opt/netronome/bin/nfp-flash --preserve-media-overrides \
    -w /opt/netronome/flash/flash-nic.bin -Z 0000:04:00.0
# /opt/netronome/bin/nfp-flash -w /opt/netronome/flash/flash-one.bin \
    -Z 0000:04:00.0
# reboot
```

Appendix F: Upgrading the Kernel¶

RHEL 7.5+¶

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

CentOS 7.5+¶

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
```

Ubuntu 18.04 LTS¶

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

Acquire packages¶

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

Install packages¶

```
dpkg -i \
    linux-headers-4.18.0-041800_4.18.0-041800.201808122131_all.deb \
    linux-headers-4.18.0-041800-generic_4.18.0-041800.201808122131_amd64.deb \
    linux-image-unsigned-4.18.0-041800-generic_4.18.0-041800.201808122131_amd64.deb \
    linux-modules-4.18.0-041800-generic_4.18.0-041800.201808122131_amd64.deb
```

Appendix G: 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.

RHEL 7.5+ and CentOS 7.5+ Grub Config¶

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

Ubuntu 18.04 LTS Grub Config¶

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

Appendix H: 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*
RHEL 7.5	`20180220-62.git6d51311.el7`
RHEL 7.6	`20180911-68.git85c5d90.el7`
RHEL 7.7	`20190429-72.gitddde598.el7`
RHEL 8.0	`20190111-92.gitd9fb2ee6.el8`
Ubuntu 18.04 LTS	`1.173`

*Netronome provides firmware packages with newer features as out-of-tree repositories. The corresponding installation packages can be obtained from Netronome Support if access to the repositories is not available. (**https://help.netronome.com** (https://help.netronome.com)).*

Installing Updated TC Firmware via the Netronome Repository¶

*Please refer to **Appendix A: Netronome Repositories** on how to configure the Netronome 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 7.5+ and CentOS 7.5+:

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

In Ubuntu 18.04 LTS:

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

Installing Updated TC Firmware from Package Installations¶

*The latest firmware can be obtained at the downloads area of the Netronome Support site (**https://help.netronome.com** (https://help.netronome.com)). Install the packages provided by Netronome Support using the commands below.*
In RHEL 7.5+ and CentOS 7.5+:

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

In Ubuntu 18.04 LTS:

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

Select Updated TC Firmware¶

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

```
# ./agilio-tc-fw-select.sh p5p1 scan flower-next
```

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

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

The initramfs image should also be update to ensure the correct firmware version is loaded at boot time.
In RHEL 7.5+ and CentOS 7.5+ this is done using:

```
# dracut -f
```

In Ubuntu 18.04 LTS use the command:

```
# update-initramfs -u
```

Appendix I: 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.

Matches¶

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

Metadata	*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*

Actions¶

A flow may be offloaded if:

1. *The input port of the flow is:*

    1. *A physical port or VF on an Agilio SmartNIC or;*
    2. *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:*

    1. *A physical port or VF on the same Agilio SmartNIC as the input port or;*
    2. *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 K: Overlay Tunneling***
Offload 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*

Appendix J: 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*

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

OVS has support for using policing to enforce a 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
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
    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 2 pkt
    backlog 0b 0p requeues 0
```

Appendix K: Overlay Tunneling¶

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.

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, something like this:

```
$ ip addr add dev (vf0_netdev) (local_vm_ip/mask)
$ ping (remote_vm_ip)
```

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`.

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:ke
```

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
...
```

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
```

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

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=(tu
```

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

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.

Appendix L: Link Aggregation (LAG)¶

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 by:

```
# 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:

```
# 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. e.g.

```
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),tc
```

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 so 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.

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.

active-backup¶

This 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.

balance-xor¶

This 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** (https://www.kernel.org/doc/Documentation/networking/bonding.txt) on the subject.*

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`.

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.

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 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.

Appendix M: 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*

Configuring QinQ in OVS¶

OVS has support to configure QinQ, previously known as 802.1ad. Support to offload this had been added in the versions above. 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 is to 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 action=NORMAL. An alternative method is to use OpenFlow rules to push and pop VLAN tags, a lot similar 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:

```
$ ovs-ofctl add-flow br0 in_port=<repr 1> action=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>,action=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 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>,action=pop_vlan,output:<repr 1>
```

There is not any 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>,action=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.

Contact Us¶

Netronome Systems, Inc. *3159 Unionville Road* *Suite 100* *Cranberry Twp., PA 16066*	
Tel: 724.778.3290	*Fax: 724.778.3295*
***https://www.netronome.com** (https://www.netronome.com)*	***help@netronome.com***