NETRONOME

Agilio OvS 2.2 User's Guide

Netronome Intelligent Server Adapters

Product code
030-00030-000

Agilio OvS 2.2 User's Guide: Netronome Intelligent Server Adapters

COPYRIGHT

WARRANTY

LIABILITY

Revision History

Date	Revision	Description
24 February 2016	000	First DRAFT for Agilio OvS 2.2.
6 April 2016	001	Added virtiorelay configuration variables, VF rate limiting, VM QoS and PXE boot feature descriptions.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 Scope

This manual provides an overview, how to configure, and how to use the Agilio OvS software.

The manual is organized as follows:

- Chapter 2 provides an overview of the Agilio OvS software architecture.
- Chapter 3 describes the configuration, usage, features and restrictions of the Agilio OvS software.
- Chapter 4 provides information on technical support.

1.2 Terminology

Acronym	Description
API	Application Programming Interface
ARP	Address Resolution Protocol
BDF	Bus/Device/Function
BSP	Board Support Package
CDP	Customer Development Platform
CPU	Central Processing Unit
DPDK	Intel© Dataplane Development Kit
ECC	Error-Correcting Code
GSG	Getting Started Guide
GUI	Graphical User Interface
LA	Link Aggregation
LACP	Link Aggregation Control Protocol
LED	Light Emitting Diode
LTS	Long Term Support
MAC	Media Access Control
MPLS	MultiProtocol Label Switching
Mpps	Millions of packets per second
MTU	Maximum Transmission Unit
NFD	Netronome Flow Driver
NFP	Netronome Network Flow Processor
NIC	Network Interface Card

Acronym	Description
NVGRE	Network Virtualization using Generic Routing Encapsulation
OvS	Open vSwitch
OS	Operating System
PCIe	Peripheral Component Interconnect Express
PMD	Poll Mode Driver
QEMU	Generic and open source machine emulator and virtualizer (http://qemu.org)
RX	Receive
SDN	Software Defined Networking
SR-IOV	Single Root I/O Virtualization
TTL	Time To Live
TX	Transmit
VF	SR-IOV Virtual Function
VirtIO	Specification for family of devices used in virtual machine environments (http://docs.oasis-open.org/virtio/virtio/v1.0/virtio-v1.0.html)
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNI	VXLAN Network Identifier
VNIC	Virtual Network Interface Card
VSID	Virtual Subnet Identifier
VXLAN	Virtual eXtensible Local Area Network

1.3 Intended Audience

This User's Guide is written for developers, integrators, and system administrators who are tasked with research, installation, configuration, integration, and testing of Netronome's Agilio OvS and Intelligent Server Adapters.

2. Overview

The Agilio OvS software provides host drivers for network connectivity to virtual machines (VM) and bare metal host applications, as well as firmware to accelerate and offload Open vSwitch (OvS) to the Netronome Intelligent Server Adapters. The Agilio OvS software runs on 64-bit x86 platforms with 64-bit Linux operating systems. In this revision, Ubuntu Server 14.04 LTS and CentOS/RHEL 7.1 distributions have been formally tested and certified by Netronome. The Agilio OvS software and firmware supports OvS version 2.5.1. The OvS kernel data path is offloaded to the Intelligent Server Adapter, improving the performance of the vSwitch as well as relieving x86 CPU cycles from vSwitch processing. Depending on workload and CPU capacity, a user can expect on the order of 5x performance gain over a non-accelerated vSwitch with CPU savings of up to 12 cores. By using OvS for the control plane, Agilio OvS can inherit traditional configuration methods present in OvS today. These include the use of ovs-ofctl utilities, as well as remote control from a controller using OVSDB for configuration and OpenFlow for flow table population. These configuration methods are also compatible with higher level software and orchestration such as OpenStack. Network connectivity to the host is supplied via Linux network devices (netdev) for terminating applications, as well as the DPDK Poll Mode Driver (PMD) for userspace applications. Both methods of host networking I/O are available to VMs as well as the host OS (a.k.a. Bare Metal Server). There are 60 virtual functions (VFs) that are available to the host or guest operating environments. The diagram in Figure 2.1 depicts a high-level view of the complete system, inclusive of Linux userspace and kernel software, VMs, drivers, and adapter firmware.

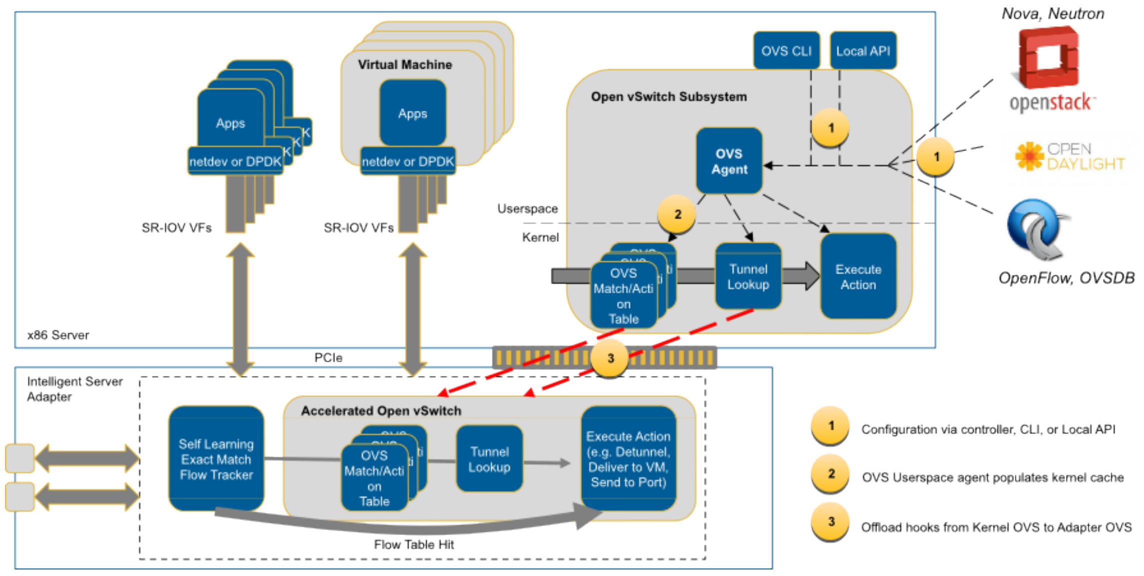

Figure 2.1. Agilio OvS Software Architecture and Interfaces

The Agilio OvS subsystem was designed to provide acceleration and offload for OvS. Therefore, configuration and integration of Agilio OvS with OvS shall be identical to an off-the-shelf, standard installation of OvS. Familiarity with OvS is a prerequisite for using Agilio OvS. In addition to this User's Guide, the OvS documentation can also be used for general OvS configuration.

2.1 Features at a Glance

The following list provides the high level features of the Agilio OvS software:

- Many Open vSwitch datapath traffic matching and action processing functions are accelerated by offload to the Netronome Intelligent Server Adapters. Where acceleration is not supported (e.g. certain control plane traffic), traffic is directed to the Open vSwitch host kernel and/or user mode code for processing in order to retain compatibility with the standard Open vSwitch behavior. This is referred to as fallback processing, and guarantees compatibility with pre-existing server networking configurations.
- Up to 60 SR-IOV VFs are available to users for each PCIe interface on the Intelligent Server Adapters. The VFs can optionally be selectively relayed into virtio connections for VMs that cannot take advantage of native SR-IOV performance.
- Accelerated Link Aggregation for a tcp-balance mode bond, with LACP enabled.
- Accelerated tunnel termination and origination, supporting VXLAN and NVGRE.
- Configurable 'wire' modes for fastpath forwarding of traffic between specified vPorts on the vSwitch.
- Traffic mirroring at ingress.
- RX and TX checksum offloading, configurable per VF.
- VM QoS: Rate limiting and network bandwidth guarantees

2.2 Supported Platforms

The following Netronome Intelligent Server Adapters are supported with this release of Agilio OvS:

- 1x40G Hydrogen - Half height, half length.
- 2x40G Starfighter - Full height, half length.
- 1x100G Starfighter - Full height, half length.
- PCIe expansion through Star Shuttle for each of the Starfighter adapters.
- 4x10G Breakout cables supported on each of the adapters supporting 40G interfaces.
- 10x10G Breakout cables supported on each of the adapters supporting 100G interfaces.
- Advantech FWA-6522C platform.
- Netronome Customer Development Platform (CDP).

2.3 Match fields

The Agilio OvS software supports many of the match fields supported by Open vSwitch. Refer to the `ovs-ofctl` manual page for details on how to use these match fields. Traffic containing fields that cannot be matched by the accelerated datapath running on the Intelligent Server Adapter is directed to the Open vSwitch host software for fallback processing.

The following match fields are offloaded to the Intelligent Server Adapter:

- Tunnel ID
- Tunnel IPv4 Source
- Tunnel IPv4 Destination
- Tunnel Flags
- Tunnel IPv4 TOS
- Tunnel IPv4 TTL
- Input port
- Ethernet source address
- Ethernet destination address
- Ethernet TCI
- Ethernet Type
- MPLS top label stack entry
- IPv4 source address
- IPv4 destination address
- IPv6 source address
- IPv6 destination address
- IPv6 flow label
- IP protocol
- IP TOS
- IP TTL
- IP Fragmentation
- Transport layer SRC
- Transport layer DST
- Transport layer flags

2.4 Actions

The following basic actions are supported by Agilio OvS and can be offloaded from the Open vSwitch kernel datapath (refer to the OvS documentation for more detailed information on the usage of these actions):

- Output to port

  This action is used to output to a specified OpenFlow port. It should be noted that this action need not be the last action in the action list. A copy of the packet will be made if this action is not the last action in the list.
- Fallback to userspace

The action used when a packet is sent to userspace. Examples of such a case could include the situation where the datapath is unable to handle a specific packet/rule combination.

- Set header

  This action sets an entire packet header. The following set actions are supported:

  - Set tunnel header

    Sets tunnel fields `tun_id`, `ipv4_src`, `ipv4_dst`, `tun_flags`, `ipv4_tos`, and `ipv4_ttl`.

  - Set ethernet header

    Sets ethernet fields `eth_src`, and `eth_dst`.

  - Set IPv4 header

    Sets IPv4 fields `ipv4_src`, `ipv4_dst`, `ip_tos`, and `ip_ttl`.

  - Set IPv6 header

    Sets IPv6 fields `ipv6_src`, `ipv6_dst`, `ip_tos`, and `ip_ttl`.

  - Set TCP header

    Sets transport layer fields `tp_src`, and `tp_dst`.

  - Set UDP header

    Sets transport layer fields `tp_src`, and `tp_dst`.

  - Set MPLS header

    Sets MPLS top label stack entry.

- Push VLAN header

  Add a VLAN header to a packet.

- Pop VLAN header

  Strip outer VLAN header from a packet.

- Push MPLS header

  Add an MPLS header to a packet.

- Pop MPLS header

  Strip outer MPLS header from a packet.

2.5 Ports

The Agilio OvS software uses the concept of representative netdevs: Linux kernel netdevs are created to represent the various ports available within the Agilio OvS system, so that these can be attached to OvS in a familiar way. Each netdev corresponds to either a physical or virtual port available in the system.

The representative netdevs process only traffic that has not been offloaded to the hardware, for example initial packets of new flows, or flows that have match fields or actions that cannot be offloaded.

The following list illustrates examples of representative netdevs:

- `sdn_p0`: representative netdev corresponding to the first physical port.
- `sdn_v0.1`: representative netdev corresponding to VF.1 on PCIe 0.
- `sdn_v1.59`: representative netdev corresponding to VF.59 on PCIe 1.

Refer to Section 3.3.1 for additional details with respect to representative netdevs.

2.5.1 Physical Ports

The Agilio OvS software supports adapters with multiple 10G, 40G, and 100G ports. Each of these physical ports will be represented by a kernel netdev of the format `sdn_pX`, where X is an integer from 0 to 47.

2.5.1.1 Configuration

On the Intelligent Server Adapter, each of the physical network interfaces can be configured to support fiber breakout cables and thereby support more lower rate physical interfaces. The physical interfaces can be configured in such a way by executing the following command:

```
/opt/netronome/bin/nfp-media phy0=CONFIG0 phy1=CONFIG1
```

where `CONFIG1` and `CONFIG2` are `4x10G`, `10x10G`, `40G`, `2x40G` or `100G` respectively. Note that `phy1=CONFIG1` should be omitted on systems with Intelligent Server Adapters that only have a single physical interface. After the physical port configuration has been modified, the host system needs to be rebooted for the changes to take effect.

The following physical network interfaces are supported on the 4xxx series Intelligent Server Adapters:

- 1x40GE
- 4x10GE

The following physical network interfaces are supported on the 6xxx series 2x40GE Intelligent Server Adapters:

- 1x40GE and 1x40GE
- 4x10GE and 1x40GE

- 4x10GE and 4x10GE

> **Note**
>
> When a mixed configuration is used (4x10GbE on one port, 1x40GbE on the other port), the highest numbered port must be the 40GbE port. phy0=1x40GE and phy1=4x10GE is not supported.

The following physical network interfaces are supported on the 6xxx series 1x100GE Intelligent Server Adapters:

- 1x100GE
- 2x40GE
- 10x10GE

The Advantech FWA-6522C platform supports combinations of up to four modules of 8x10G, 2x40G or 1x100G modules respectively.

2.5.1.2 LED Management

Please refer to the Hardware User Guide for pictures and LEDs-to-port mapping as this depends on the specific Intelligent Server Adapter used.

The following LEDs are used in the Intelligent Server Adapters:

- Activity LED (Yellow): Blinks on activity and is updated every 100ms.
- Link LED (Green): Active when MAC detects link from its peer.

> **Note**
>
> When a port is configured to be used with a breakout cable, e.g. a 4x10G port configuration on a 40G interface, the LEDs will only be updated for the first port in the set of ports on the breakout cable. This is applicable to all Intelligent Server Adapters that support breakout cables.

2.5.2 Virtual Function (VF) Ports

60 VFs per physical function are available to the user. The number of physical functions (PCIe units) is dependent on the platform. These VFs can be bound to DPDK applications (i.e. Poll Mode Drivers running in userspace), or to kernel netdevs. The DPDK applications and netdevs can be deployed in VMs or can be instantiated on the host itself. The VFs are represented by host kernel representative netdevs named `sdn_vX.Y`, where X is the NFP PCIe unit number (0-3) and Y is the VF number on that PCIe unit. It is important to understand that these kernel representative netdevs are NOT the VF traffic endpoints themselves. They merely provide a representative netdev that can be connected to the Open vSwitch software running on the host to

facilitate non-accelerated processing of traffic. They can furthermore be used to obtain traffic statistics of each of the fallback and fastpathed traffic.

Note
Only 60 VFs per PCIe unit are available to the user. VFs 60 to 63 are reserved.

2.5.3 OpenFlow Ports

Any representative netdev, either a physical port netdev or a VF netdev, may be added to an Open vSwitch bridge[1]. The OpenFlow port configuration requested by the user when configuring Open vSwitch will be honored with respect to all Open vSwitch commands. The OpenFlow port number may thus be used as an user configurable port number.

2.6 Link Aggregation

Link Aggregation (also known as bonding, port trunking or link bundling) allows using multiple network connections in parallel to:

- Provide redundancy (active / backup links)
- Provide increased throughput (active / active links)

In the case of increased throughput, for each packet sent to the bond, one output port is selected from the set of ports included in the bond.

The accelerated implementation of Link Aggregation supports 32 independent bonds, each containing up to 128 ports, using the tcp-balance bond mode. Each port in a bond is selected with equal likelihood. A bond can contain a combination of physical ports and VFs. LACP (Link Aggregation Control Protocol) is supported, in both active / active and active / passive modes.

2.7 Counters

Agilio OvS software supports all counters made available by Open vSwitch. Refer to the appropriate Open vSwitch documentation for instructions w.r.t. how to extract counters from the system. The software ensures that the Open vSwitch counters maintained on the host accurately reflect all traffic, whether traffic was handled by the accelerated datapath on the NFP or whether it was handed to the Open vSwitch software on the host for fallback processing.

[1]Any 3rd party NIC netdev or tap netdev may also be added to an OvS bridge. OpenFlow flows containing match fields or actions related to such ports will simply not be offloaded to the NFP, but will still function correctly.

2.8 Checksum Offload

Agilio OvS software supports L3 and L4 checksum offload on all VF ports. Both the Linux kernel network driver and the Poll Mode (DPDK) drivers are supported. Furthermore RX and TX checksum offloading is supported for traffic flowing from the network to a VF, from a VF to the network and from a VF to a VF. Layer 3 protocols IPv4 and IPv6 are supported. Layer 4 protocols TCP and UDP are supported.

The IP header checksum is checked as per RFC 791, where the packet meets the following criteria:

- If present, the DSA-tag header must be four or 8 octets long.
- If present, the VLAN header must be four octets long.
- Encapsulation must be RFC 894 Ethernet Type Encoding.
- IPv4 packet - (ethertype == 0x800) AND (ver == 4) AND (hdr_len >= 5).
- IP header is of a valid length.
- The packet can have at most 2 levels of VLAN tags.
- The packet can have at most 8 levels of MPLS tags.
- If both MPLS and VLAN are present, the packet can have at most 8 MPLS and 2 VLAN tags.
- The layer 3 protocols that are supported are IPv4 and IPv6.

The TCP/UDP header checksum is calculated as per RFC 793, where the packet meets the following criteria:

- IPv4 or IPv6.
- No IP fragmentation.
- No ESP header.
- No Authorization header.
- No Mobility header.
- TCP or UDP packet.

2.9 VM QoS

Agilio OvS software supports setting the maximum bit rate a VM(Virtual Machine) can send or receive. This is enforced on the VF(s) (Virtual Functions) that are bound to the VM, It also supports the ability for a VM to reserve bandwidth when transmitting on a physical network port.

3. Functionality and Interfaces

3.1 Configuration

The Agilio OvS software can be configured by using the `/etc/netronome.conf` file. This file takes the form of `variable=value` entries, one per line. Lines starting with the "#" character are treated as comments and are ignored. The table below lists the variables that are recognized in this file. For each variable, if applicable, the default value and the permissible values or the acceptable value range are specified.

Table 3.1. Agilio OvS Configuration Variables

Name / Description	Unit	Valid values	Default
SDN_VERBOSE: Determines whether to perform verbose logging. This only affects the logging performed when the software is started and stopped.	Text	"y" or "n"	"n"
SDN_VIRTIORELAY_2M_HUGEPAGES: Selects the number of 2Mb hugepages which will be configured automatically when starting Agilio OvS. The startup procedure will only increase the number of hugepages if the currently configured number of hugepages is less than the specified number. Setting the number of hugepages to 0, prevents the startup script from making any modification to the hugepage configuration on the system.	Integer	0 - available memory on host system	768
SDN_VIRTIORELAY_ENABLE: Enables or disables the virtiorelayd process from starting up along with Agilio OVS.	Text	"y" or "n"	"n"
SDN_VIRTIORELAY_PARAM: Provides the user the ability to modify the commandline configuration options to the virtiorelayd process. This variable is set to the default value during installation unless there is already an existing value defined.	Text	Refer to virtiorelayd usage.	"--cpus=1,2 --vhost-username=libvirt-qemu --vhost-groupname=kvm --huge-dir=/mnt/aovs-huge-2M"

None of the variables are mandatory. If they are not specified the system will use the respective default value.

The following example illustrates a valid configuration file.

```
SDN_VERBOSE=n
SDN_VIRTIORELAY_2M_HUGEPAGES=128
```

3.2 ovs-ctl Command Usage

The `ovs-ctl` command is used to control the Agilio OvS software. The Agilio OvS software is directly hooked into Open vSwitch, therefore the `ovs-ctl` command used to manage the Open vSwitch software will also manage the Agilio OvS software. Refer to the manual page of the `ovs-ctl` for all the details.

The following sections provide more detail on the specific Netronome extensions to `ovs-ctl`, along with examples of the usage of the particular command and expected output. The examples below are not indicative of the complete scope of all inputs and outputs, but merely illustrate the normal usage of the command.

3.2.1 Main Startup/Shutdown Command for Agilio OvS Software and Firmware

- **Command:**

  ```
  /opt/netronome/bin/ovs-ctl [start|stop|restart]
  ```

- **Syntax Description:**

Parameter	Description
`start`	Start the Agilio OvS host software (kernel modules and daemons) as well as the Network Flow Processor firmware. This will also start Open vSwitch software.
`stop`	Stop the Agilio OvS host software and firmware. Kernel modules no longer required will be unloaded. This will also stop Open vSwitch software.
`restart`	Similar to `ovs-ctl stop` followed by `ovs-ctl start`, except that the OpenFlow flows are preserved.

- **Command Mode:**

  Privileged Execution

- **Usage Guidelines:**

  It is recommended that no other commands be executed while waiting for the Agilio OvS software to start or stop.

  After the system has been started, the following processes should be running for normal system operation:

  - `ovs-vswitchd`
  - `ovsdb-server`
  - `virtiorelayd`

  The required processes and their current status can be viewed with the `ovs-ctl status` command. This command is described in more detail below.

The `ovs-ctl stop` will not execute when there are any DPDK applications or VMs connected to any VFs. A warning with the respective process which is still running will be reported to the user.

All options provided by the standard OvS `ovs-ctl` are available, for example to prevent the following log message

```
 * system ID not configured, please use --system-id
```

one can specify `ovs-ctl start --system-id=random`. Refer to the man page for the standard OvS options and parameters applicable to the `ovs-ctl`.

- **Examples:**

  Examples of typical commands and the expected output are supplied below. There may be some minor variation of the output depending on the platform the software is used on.

  Command:

  ```
  /opt/netronome/bin/ovs-ctl start
  ```

  Output:

  ```
   * Starting Agilio OvS 2.2
   * Resetting NFP, please wait...
   * Starting ovsdb-server
   * system ID not configured, please use --system-id
   * Configuring Open vSwitch system IDs
   * Starting ovs-vswitchd
   * Enabling remote OVSDB managers
   * Agilio OvS 2.2 started successfully
   * Running firmware image /lib/firmware/nfp_firmware2.2_648x.nffw
  ```

  Command:

  ```
  /opt/netronome/bin/ovs-ctl stop
  ```

  Output:

  ```
   * Stopping Agilio OvS 2.2
   * Exiting ovs-vswitchd (70074)
   * Exiting ovsdb-server (70003)
   * Resetting NFP, please wait...
   * Agilio OvS 2.2 stopped successfully
  ```

3.2.2 Display Agilio OvS Software Version

- **Command:**

  ```
  /opt/netronome/bin/ovs-ctl version
  ```

- **Syntax Description:**

Parameter	Description
`version`	Display the current version of the Agilio OvS software installed on the system.

- **Command Mode:**

  User Execution

- **Usage Guidelines:**

  This version text should accompany any queries logged with Netronome support.

- **Example:**

  Examples of typical commands and the expected output are supplied below.

  Command:

```
/opt/netronome/bin/ovs-ctl version
```

  Output:

```
ovsdb-server (Open vSwitch) 2.5.1
Compiled Apr  5 2016 21:16:57
ovs-vswitchd (Open vSwitch) 2.5.1
Compiled Apr  5 2016 21:17:04
Netronome Agilio OvS version 2.2-r3945:fe8efa9ee970
NFP Firmware Loaded: /lib/firmware/nfp_firmware2.2_648x.nffw
```

3.2.3 Display Agilio OvS System Health Status

- **Command:**

```
/opt/netronome/bin/ovs-ctl status
```

- **Syntax Description:**

Parameter	Description
`status`	Display the current health status of the Agilio OvS software and hardware.

- **Command Mode:**

  Privileged Execution

- **Usage Guidelines:**

  The health monitor utility will display the information for the NFP installed in the system and the current software status.

This command can be invoked routinely or whenever there is a suspicion that the system may have failed. It is recommended that this command should not be invoked while starting or stopping the Agilio OvS software.

Invoking this command will log any failures detected to syslog. Refer to Section 3.8 for the exact messages that will be logged to syslog for each type of failure. These syslog messages also describe the recommended action to take to resolve the failure.

If the syslog messages contain "restart", e.g. "restart.process", "restart.drivers", etc., restart the system by entering the command `/opt/netronome/bin/ovs-ctl restart`. If the problem occurs again, power cycle the system. If the problem persists, enter the command `/opt/netronome/bin/ovs-ctl status troubleshoot` and contact Netronome support. Similarly, if the syslog messages contain "replace", e.g. "replace.nfe", the hardware may need to be replaced by contacting Netronome support.

The return code of this command indicates the number of failures detected. Warnings could also be logged, but although these warnings must be brought to the attention of the user, the system is not in a failed state and the return code of the command is zero.

- **Examples:**

Examples of typical commands and the expected output are supplied below.

Command: `/opt/netronome/bin/ovs-ctl status` on a system currently running Agilio OvS software.

Output:

```
     Detected Configurator version: 20151110145024

SDN Health Report:
------------------
Userspace Process: ovsdb-server              ... [PASS]
Userspace Process: ovs-vswitchd              ... [PASS]
Userspace Process: virtiorelayd              ... [PASS]
Kernel Module: nfp                           ... [PASS]
Kernel Module: nfp_cmsg                      ... [PASS]
Kernel Module: nfp_fallback                  ... [PASS]
Kernel Module: nfp_offloads                  ... [PASS]
Kernel Module: openvswitch                   ... [PASS]
NFP: Firmware Loaded                         ... [PASS]
NFP: Flow Processing Cores Responsive        ... [PASS]
NFP: Control Message Channel Responsive      ... [PASS]
NFP: Ingress NBI Backed Up                   ... [PASS]
NFP: Card Detected                           ... [PASS]
NFP: ECC Errors                              ... [PASS]
NFP: Parity Errors                           ... [PASS]
NFP: Configurator Version                    ... [PASS]
Host: ERR47 Workaround                       ... [PASS]

Overall System State     [PASS]

virtiorelayd is running with pid 70744
ovsdb-server is running with pid 70003
ovs-vswitchd is running with pid 70074
```

Command: `/opt/netronome/bin/ovs-ctl status` on a system with Agilio OvS software currently stopped.

Output:

```
     Detected Configurator version: 20151110145024

SDN Health Report:
------------------
Userspace Process: ovsdb-server             ... [FAIL]
Userspace Process: ovs-vswitchd             ... [FAIL]
Userspace Process: virtiorelayd             ... [FAIL]
Kernel Module: nfp                          ... [PASS]
Kernel Module: nfp_cmsg                     ... [FAIL]
Kernel Module: nfp_fallback                 ... [FAIL]
Kernel Module: nfp_offloads                 ... [FAIL]
Kernel Module: openvswitch                  ... [PASS]
NFP: Firmware Loaded                        ... [FAIL]
NFP: Flow Processing Cores Responsive       ... [WARN]
NFP: Control Message Channel Responsive     ... [FAIL]
NFP: Ingress NBI Backed Up                  ... [PASS]
NFP: Card Detected                          ... [PASS]
NFP: ECC Errors                             ... [PASS]
NFP: Parity Errors                          ... [PASS]
NFP: Configurator Version                   ... [PASS]
Host: ERR47 Workaround                      ... [PASS]

Overall System State     [FAIL]
```

3.2.4 Gather System Troubleshooting Information

- **Command:**

  `/opt/netronome/bin/ovs-ctl status troubleshoot`

- **Syntax Description:**

Parameter	Description
`troubleshoot`	Compiles and archives various troubleshooting information about the system which should be supplied to Netronome support when system diagnostics are required.

- **Command Mode:**

  Privileged Execution

- **Usage Guidelines:**

  The troubleshoot utility will gather information of the system components and archive this in a single file stored in the `/opt/netronome/sysinfo/archives/` directory.

  The troubleshoot utility should be invoked as soon as possible after a failure has been observed to obtain the most usable data for diagnostics later on.

While this command is executing, each of the component groups is displayed while information is extracted from it. The only important information displayed to the user is the name and location of the archive file where the collected data will be stored. The messages displayed on the screen merely indicate progress as this command may take a long time to process, but these are of no importance to the user of system even if failures are reported.

If an error occurs while trying to extract information from the system component, the "done" status printed at end of each line will be replaced with a "failed" message. If this does occur, some of the debugging information contained within the archive may not be complete but it may also be a consequence of the actual system failure. The remainder of the information will still be available.

Note

After the execution of the troubleshoot utility the host system will need to be rebooted due to the destructive nature of gathering the debugging data.

- **Example:**

  Examples of typical commands and the expected output are supplied below.

  Command:

```
/opt/netronome/bin/ovs-ctl status troubleshoot
```

  Output:

```
Netronome Systems - System Troubleshoot Information

Saving last logs... done
Saving kernel ring buffer (dmesg)... done
Saving device info (lspci)... done
Saving NFE HWINFO and SDN config files...done
Saving versions... done
Saving status of SDN components... done
Saving information about host system... done
Saving `top' util output... done
Saving current process information... done
Saving ARP/Routing tables... done
Saving ME Information (this may take a while)... done
Saving current OVS configuration information... done
Saving flow entries... done
Saving flow cache diagnostics (this may take up to 10 minutes)... done
Saving NFD info...done
Saving control channel info...done
Saving GRO dump...done
Saving Port Information... done
Saving NFP symbols (this may take a while)... done
Saving NFP Debug Information (this may take up to 5 minutes)... done
Saving symbol list dump...done
Saving worker states...done
Saving Error Correct Code info... done
Dumping QINFO ... done
Dumping GPRs and Local Memory ... done
```

```
Archiving system information... done

Please review the contents of /opt/netronome/sysinfo and e-mail
/opt/netronome/sysinfo/archives/sysinfo-2016-04-06.1053.tgz
to support@netronome.com along with a description of the problem you have
encountered. If possible, also provide steps to reproduce the problem.

Please reboot the system. Due to the troubleshoot information gathered,
the system will need to be rebooted to operate normally again.
```

3.2.5 Fast Path (Wire) Configuration

- **Commands:**

  `/opt/netronome/bin/ovs-ctl configure wire from IN_PORT to OUT_PORT`

  `/opt/netronome/bin/ovs-ctl configure wire clear IN_PORT`

- **Syntax Description:**

Parameter	Description
`IN_PORT`, `OUT_PORT`	The PORT parameters refer to representative netdevs as described in Section 3.3.1.

- **Command Mode:**

  Privileged Execution

- **Usage Guidelines:**

  There are no specific usage guidelines for this command.

- **Example:**

  Examples of typical commands are supplied below.

  To bidirectionally connect physical ports sdn_p0 and sdn_p1:

```
ovs-ctl configure wire from sdn_p0 to sdn_p1
ovs-ctl configure wire from sdn_p1 to sdn_p0
```

  To bidirectionally connect physical port sdn_p1 to the DPDK application / netdev attached to VF 1 on NFP PCIe unit 0:

```
ovs-ctl configure wire from sdn_p1 to sdn_v0.1
ovs-ctl configure wire from sdn_v0.1 to sdn_p1
```

  To remove a wire (i.e. resume normal operations):

```
ovs-ctl configure wire clear sdn_p1
ovs-ctl configure wire clear sdn_v0.1
```

3.2.6 Tap/Mirror Configuration

- **Commands:**

  ```
  /opt/netronome/bin/ovs-ctl configure mirror rx from IN_PORT to OUT_PORT

  /opt/netronome/bin/ovs-ctl configure mirror rx clear IN_PORT
  ```

- **Syntax Description:**

Parameter	Description
`IN_PORT`, `OUT_PORT`	The PORT parameters refer to representative netdevs as described in Section 3.3.1.

- **Command Mode:**

  Privileged Execution

- **Usage Guidelines:**

  Configure a tap or mirror on a physical port or a virtual function. Packet will be copied on reception and the copy will be sent to the mirror destination. Expect a performance impact under high packet rates.

- **Example:**

  Examples of typical commands are supplied below.

  To mirror packets from physical ports sdn_p0 to sdn_p1:

  ```
  ovs-ctl configure mirror rx from sdn_p0 to sdn_p1
  ```

  To mirror physical port sdn_p0 to the DPDK application / netdev attached to VF 1 on NFP PCIe unit 0:

  ```
  ovs-ctl configure mirror rx from sdn_p0 to sdn_v0.1
  ```

  To remove a mirror (i.e. resume normal operations):

  ```
  ovs-ctl configure mirror rx clear sdn_p0
  ovs-ctl configure mirror rx clear sdn_v0.1
  ```

3.3 Configuring Open vSwitch

3.3.1 Representative Netdev

The Agilio OvS software uses the concept of representative netdevs. Kernel netdevs are automatically created by the `nfp_fallback.ko` driver, and represent the physical and virtual function interfaces. When these netdevs are connected to an Open vSwitch bridge, traffic on them will automatically trigger the offloading of flows to the NFP. Examples of representative netdevs are:

- `sdn_p0`: representative netdev corresponding to the first physical port.
- `sdn_v0.1`: representative netdev corresponding to VF.1 on NFP PCIe unit 0.
- `sdn_v1.59`: representative netdev corresponding to VF.59 on NFP PCIe unit 1.

Offloading flows to the NFP eventually results in fewer fallback packets being emitted from the representative netdevs. Running `ethtool -S` on these representative netdevs will give two distinct sets of packet and byte counters: one for the underlying device (physical port or VF), and one set for the actual fallback traffic received on the host representative netdev. Refer to the table below for a complete reference of all the port statistics supported by the software. The fallback traffic counters will ordinarily be far less than the underlying device counters when flows have been successfully offloaded to the NFP. The counters displayed by `ifconfig` (also seen in OvS port stats) represent the underlying device, not the fallback traffic.

Virtual Function devices can have two netdevs associated with them. The representative netdev representing the VF, and the netdev running on the actual VF (Refer to Section 3.6.1). The important distinction is that the representative netdev should only emit packets from flows that have not been offloaded to the NFP. The netdev running directly on the VF will receive and transmit all packets: fallback packets and offloaded packets will both be routed correctly. The actual VF can of course also be assigned to a VM, and/or be used by a DPDK poll mode driver instead of a kernel netdev driver, and the VF is where user applications will receive and transmit traffic. The VF netdev is optimized for bulk traffic performance, whereas the representative netdevs are considered to be a low bandwidth traffic path.

Consequently, to allow for proper offloading, the representative device is added to an OvS bridge and the netdev (or DPDK PMD, or VM etc.) associated with the VF is where user application traffic is directed.

The netdev running directly on the VF is named by the Linux distribution. The Agilio OvS software does not override this and the standard rules implemented by `udev`, for example, are applied.

The following table provides detailed descriptions of all the port statistics supported by physical ports on the NFP. These statistics are available via the ethtool utility. VF netdevs will provide a subset of these statistics.

Table 3.2. Port counters available via ethtool

Counter	Description
`fallback rx bytes`	Bytes received by representative netdev.
`fallback rx packets`	Packets received by representative netdev.
`fallback tx bytes`	Bytes transmitted by representative netdev.
`fallback tx packets`	Packets transmitted by representative netdev.
`fallback tx errors`	Transmit errors detected by representative netdev.
`fallback tx copies`	Packets copied on transmission by representative netdev.
`device rx bytes`	See description of `RxPIfInOctets` below.
`device rx packets`	See description of `RxFramesReceivedOK` below.
`device tx bytes`	See description of `TxPIfOutOctets` below.
`device tx packets`	See description of `TxFramesTransmittedOK` below.
`device rx frame errors`	See description of `RxFrameCheckSequenceErrors` below.

Counter	Description
`device rx errors`	See description of `RxPIfInErrors` below.
`device rx multicast`	See description of `RxPIfInMultiCastPkts` below.
`device rx dropped`	See description of `RxPStatsDropEvents` below.
`device tx errors`	See description of `TxPIfOutErrors` below.
`RxAlignmentErrors`	Frames received with an alignment error.
`RxCBFCPauseFramesReceived0`	CBFC (Class Based Flow Control) pause frames received for class 0.
`RxCBFCPauseFramesReceived1`	CBFC (Class Based Flow Control) pause frames received for class 1.
`RxCBFCPauseFramesReceived2`	CBFC (Class Based Flow Control) pause frames received for class 2.
`RxCBFCPauseFramesReceived3`	CBFC (Class Based Flow Control) pause frames received for class 3.
`RxCBFCPauseFramesReceived4`	CBFC (Class Based Flow Control) pause frames received for class 4.
`RxCBFCPauseFramesReceived5`	CBFC (Class Based Flow Control) pause frames received for class 5.
`RxCBFCPauseFramesReceived6`	CBFC (Class Based Flow Control) pause frames received for class 6.
`RxCBFCPauseFramesReceived7`	CBFC (Class Based Flow Control) pause frames received for class 7.
`RxFrameCheckSequenceErrors`	CRC-32 Error is detected but the frame is otherwise of correct length.
`RxFrameTooLongErrors`	Frame received exceeded the maximum length.
`RxFramesReceivedOK`	Frame received without error (including pause frames).
`RxInRangeLengthErrors`	A count of frames with a length / type field value between 46 (VLAN: 42) and less than 0x0600, that does not match the number of payload data octets received. Should count also if length / type field is less than 46 (VLAN: 42) and the frame is longer than 64 bytes.
`RxPIfInBroadCastPkts`	Incremented with each valid frame received on the Receive FIFO interface and all bits of the destination address were set to '1'.
`RxPIfInErrors`	Number of frames received with error: FIFO Overflow Error, CRC Error, Frame Too Long Error, Alignment Error, The dedicated Error Code (0xfe, not a code error) was received.
`RxPIfInMultiCastPkts`	Incremented with each valid frame received on the Receive FIFO interface and bit 0 of the destination address was '1' but not the broadcast address (all bits set to '1'). Pause frames are not counted.
`RxPIfInUniCastPkts`	Incremented with each valid frame received on the Receive FIFO interface and bit 0 of the destination address was '0'.
`RxPStatsDropEvents`	Counts the number of dropped packets due to internal errors of the MAC Client. Occurs when a Receive FIFO overflow condition persists.
`RxPStatsFragments`	Total number of packets that were less than 64 octets long with a wrong CRC. Note: Fragments are not delivered to the FIFO interface.
`RxPStatsJabbers`	Total number of packets longer than the valid maximum length programmed in register FRM_LENGTH (excluding framing bits, but including FCS octets), and with a bad Frame Check Sequence.

Counter	Description
`RxPStatsOversizePkts`	Total number of packets longer than the valid maximum length programmed in register FRM_LENGTH (excluding framing bits, but including FCS octets), and with a good Frame Check Sequence.
`RxPStatsPkts`	Total number of packets received. Good and bad packets.
`RxPStatsPkts1024to1518octets`	Frames (good and bad) with 1024 to 1518 octets.
`RxPStatsPkts128to255octets`	Frames (good and bad) with 128 to 255 octets.
`RxPStatsPkts1519toMaxoctets`	Proprietary RMON extension counter that counts the number of frames with 1519 bytes to the maximum length programmed in register FRM_LENGTH.
`RxPStatsPkts256to511octets`	Frames (good and bad) with 256 to 511 octets.
`RxPStatsPkts512to1023octets`	Frames (good and bad) with 512 to 1023 octets.
`RxPStatsPkts64octets`	Incremented when a packet of 64 octets length is received (good and bad frames are counted).
`RxPStatsPkts65to127octets`	Frames (good and bad) with 65 to 127 octets.
`RxPStatsUndersizePkts`	Total number of packets that were less than 64 octets long with a good CRC. Note: Undersized packets are not delivered to the FIFO interface.
`RxPauseMacCtlFramesReceived`	Valid pause frame received.
`RxVlanReceivedOK`	VLAN frame received without error.
`TxCBFCPauseFramesTransmitted0`	CBFC (Class Based Flow Control) pause frames transmitted for class 0.
`TxCBFCPauseFramesTransmitted1`	CBFC (Class Based Flow Control) pause frames transmitted for class 1.
`TxCBFCPauseFramesTransmitted2`	CBFC (Class Based Flow Control) pause frames transmitted for class 2.
`TxCBFCPauseFramesTransmitted3`	CBFC (Class Based Flow Control) pause frames transmitted for class 3.
`TxCBFCPauseFramesTransmitted4`	CBFC (Class Based Flow Control) pause frames transmitted for class 4.
`TxCBFCPauseFramesTransmitted5`	CBFC (Class Based Flow Control) pause frames transmitted for class 5.
`TxCBFCPauseFramesTransmitted6`	CBFC (Class Based Flow Control) pause frames transmitted for class 6.
`TxCBFCPauseFramesTransmitted7`	CBFC (Class Based Flow Control) pause frames transmitted for class 7.
`TxFramesTransmittedOK`	Frame transmitted without error (including pause frames).
`TxPIfOutBroadCastPkts`	Incremented with each frame written to the FIFO interface and all bits of the destination address set to '1'.

Counter	Description
`TxPIfOutErrors`	Number of frames transmitted with error: FIFO Overflow Error, FIFO Underflow Error, or User application defined error (ff_tx_err asserted together with ff_tx_eop).
`TxPIfOutMultiCastPkts`	Incremented with each frame written to the FIFO interface and bit 0 of the destination address set to '1' but not the broadcast address (all bits set to '1').
`TxPIfOutUniCastPkts`	Incremented with each frame written to the FIFO interface and bit 0 of the destination address set to '0'.
`TxPStatsPkts1024to1518octets`	Frames (good and bad) with 1024 to 1518 octets.
`TxPStatsPkts128to255octets`	Frames (good and bad) with 128 to 255 octets.
`TxPStatsPkts1518toMAXoctets`	Frames (good and bad) with 1518 to MAX octets.
`TxPStatsPkts256to511octets`	Frames (good and bad) with 256 to 511 octets.
`TxPStatsPkts512to1023octets`	Frames (good and bad) with 512 to 1023 octets.
`TxPStatsPkts64octets`	Incremented when a packet of 64 octets length is received (good and bad frames are counted).
`TxPStatsPkts65to127octets`	Frames (good and bad) with 65 to 127 octets.
`TxPauseMacCtlFramesTransmitted`	Valid pause frame transmitted.
`TxVlanTransmittedOK`	VLAN frame transmitted without error.
`RxPIfInOctets`	All octets received except preamble (i.e. Header, Payload, Padding and FCS) for all frames and pause frames received.
`TxPIfOutOctets`	All octets transmitted except preamble (i.e. Header, Payload, Padding and FCS) for all valid frames and valid pause frames transmitted.

3.3.2 Example Bridge Setup

Assuming `br0` for these examples, create the bridge:

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

Set the bridge OpenFlow protocol version by entering on the host, logged in as root:

```
ovs-vsctl set Bridge br0 protocols=OpenFlow13
```

and add ports to the bridge specifying the OpenFlow port numbers by using:

```
ovs-vsctl add-port br0 $PORT -- set Interface $PORT ofport_request=$PORT_NUM
```

`$PORT`, in this case, corresponds to any representative netdev associated with the NFP. These are generally in the form `sdn_pX` for physical ports, or `sdn_vY.Z` for Virtual Functions. `$PORT_NUM` is the OpenFlow number that is requested to be associated with that port, and is a user configurable integer with a minimum value of 1. This OpenFlow port number is used when specifying ingress port match field or output actions with OpenFlow flows.

The OpenFlow/Port mapping can be inspected by using:

```
ovs-ofctl -OOpenFlow13 show br0
```

Setting the OpenFlow protocol version of the bridge implies that OpenFlow operations such as those performed with the `ovs-ofctl` utility must be performed with compatible OpenFlow versions, for example adding flow entries:

```
ovs-ofctl -OOpenFlow13 add-flow br0 in_port=$INPORT,actions=output:$PORT_NUM
```

Set the bridge failure mode (behaviour when connection to the controller is lost). By default (usually at startup) Open vSwitch enters standalone failure mode. This may be changed by using:

```
ovs-vsctl set-fail-mode br0 secure
```

Set the datapath maximum idle timeout by using:

```
ovs-vsctl set Open_vSwitch . other_config:max-idle=$TIMEOUT
```

Setting `TIMEOUT=-1` (specified in milliseconds) is interpreted as infinity. The default timeout is 1500ms (1.5s), it can be handy to increase this timeout (or set to infinity) when trying to inspect datapath flows (`ovs-dpctl dump-flows`) to ensure they are not timed out before you have a chance to see them.

3.3.3 Layer 2 - Learning Bridge

Configure Open vSwitch (explicitly) to act as a learning bridge by using the following command:

```
ovs-ofctl -OOpenFlow13 add-flow br0 actions=NORMAL
```

In Open vSwitch this is simply a flow rule (not strictly a slice of the switch resources) and thus may be combined with other flow rules in the OpenFlow pipeline. The flow rule with action NORMAL is installed by default when the failure mode is standalone and no OpenFlow controller is connected.

3.3.4 Layer 3 - Routing

Matching on L3 addresses and then modifying L2 addresses as well as TTL/Hop count results in datapath actions that set both the L2 and L3 headers, for example:

```
ovs-ofctl -OOpenFlow13 add-flow br0 \
ip,nw_dst=192.168.1.0/24,actions=set_field:00:1a:2b:3c:4d:5e-\>dl_src,\
set_field:00:11:aa:22:bb:33-\>dl_dst,dec_ttl,output:2
```

This rule matches the 192.168.1.0/24 subnet, sets the source and destination ethernet addresses and decrements the IP TTL before outputting to port 2.

3.3.5 Tags and Labels

Adding and removing of tags/labels may be achieved as follows using flow rules.

3.3.5.1 VLAN Tags

To push a VLAN tag and set the VID and PCP:

```
ovs-ofctl -OOpenFlow13 add-flow br0 actions=push_vlan:0x8100,set_field:4196-\>\
vlan_vid,set_field:3-\>vlan_pcp,output:2
```

Note
Only an ethertype of 0x8100 is supported and therefore only a single VLAN tag may be pushed onto a packet.

In order to pop a VLAN tag from a packet use the following:

```
ovs-ofctl -OOpenFlow13 add-flow br0 dl_vlan=1,actions=strip_vlan,output:2
```

3.3.5.2 MPLS Labels

Pushing MPLS labels can be done using the following flow rule, which simultaneously sets the label, TC and TTL:

```
ovs-ofctl -OOpenFlow13 add-flow br0 actions=push_mpls:$ETHERTYPE,set_field:100-\>\
mpls_label,set_field:3-\>mpls_tc,set_mpls_ttl:32,output:2
```

`$ETHERTYPE` may be one of either `0x8847` or `0x8848` for unicast or multicast MPLS respectively.

Removing MPLS labels is much the same as popping VLAN tags:

```
ovs-ofctl -OOpenFlow13 add-flow br0 dl_type=$ETHERTYPE_MATCH,actions=pop_mpls:\
$ETHERTYPE_OUT,output:2
```

however `$ETHERTYPE_OUT` in this case specifies the ethernet type of the resulting packet and `$ETHERTYPE_MATCH` the ethernet type of the ingress packet to be matched.

3.3.6 Running Packet Analyzer Tools (tcpdump) on Representative Netdevs

Due to the architecture of the software, one cannot always run packet analyzer tools like tcpdump on a representative netdev. As soon as all the flows are offloaded to the NFP, the traffic may no longer traverse the host at all. An example of this is simple match port/set port rules. All traffic will be forwarded to the opposite port, and only stats updates will reach the host. In this case, something like `tcpdump -i sdn_p0` will not show any traffic.

To effectively run analysis tools on a representative netdev in this case, one needs to add a tap/mirror on the physical port to send traffic to an unused VF:

```
ovs-ctl configure mirror rx from sdn_p0 to sdn_v0.1
```

Now run tcpdump on the netdev attached to VF 1 on NFP PCIe unit 0.

For more information on configuring mirrors, please refer to Section 3.2.6

3.4 Tunneling

The Agilio OvS software supports accelerated termination and origination of VXLAN and NVGRE tunnels. The Agilio OvS tunnel configuration is the same as configuring tunnels with unaccelerated Open vSwitch.

Open vSwitch provides a multitude of methods to configure tunnel endpoints, however to make full use of the acceleration of tunnels with Agilio OvS software, the following configuration should be used.

The port on which tunneled traffic ingresses and egresses the system should not be added to the Open vSwitch bridge. Instead, this port (or ports) must be configured with the tunnel endpoint IP address using Linux system tools like `ifconfig` or `ip`. Any other system ports should be added to Open vSwitch and a tunnel configuration created in the Open vSwitch bridge.

The following commands can be used to configure such a tunnel. This example assumes that entunneled traffic will egress the system on netdev `sdn_p1` to a remote peer with IP address `10.1.1.2`. The rules are configured in such a manner that all traffic ingressing from `sdn_p0` will be entunneled and sent out of `sdn_p1`. Conversely, all entunneled traffic ingressing on `sdn_p1`, will be detunneled and sent out of `sdn_p0` again. The `sdn_p1` netdev is not added to the Open vSwitch bridge since it is configured with the remote IP address of the tunnel endpoint. This example assumes that there are two physical ports available. The example can be modified to use a single physical port by changing all references to `sdn_p0` to a VF port and configuring the single physical port in the same manner as the `sdn_p1` netdev in this example.

```
ovs-vsctl add-br br0
ovs-vsctl add-port br0 sdn_p0 -- set interface sdn_p0 ofport_request=1
ovs-vsctl add-port br0 tun0 -- set Interface tun0 type=vxlan options:remote_ip=10.1.1.2 \
options:local_ip=10.1.1.1 options:key=flow ofport_request=2
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=1,actions=set_tunnel:0xff,output:2
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=2,actions=1
ifconfig sdn_p1 10.1.1.1/24 up
```

For more information about the Open vSwitch commands used in this example configuration, refer to the Open vSwitch documentation and manual pages for the respective utilities.

The Agilio OvS software does support header modification actions along with tunnels to the same extent as supported by Open vSwitch. The Agilio OvS software also supports tunnel conversion over all permutations of VXLAN and NVGRE. For example, VXLAN entunneled traffic can be detunneled and entunneled again using NVGRE.

3.5 Link Aggregation

The Agilio OvS software supports accelerated Link Aggregation, using the tcp-balance bond mode, to up to 32 independent bonds, where each bond consists of up to 128 ports, where each port can be a physical port or a VF.

A bond is typically created between two machines that are connected via multiple physical links. To configure each machine, proceed as follows:

To associate a specific port with a bond, the OpenFlow port bound to the representative netdev is used. Refer to Section 3.3.1 for more information on the representative netdevs.

To create a bond (named `bondbr0` in this example) over four physical ports (`sdn_p0`, `sdn_p1`, `sdn_p2` and `sdn_p3`) to provide redundancy (active / backup links) where one link is used, and another is selected when the current link fails (default behaviour):

```
ovs-vsctl add-bond bondbr0 bond0 sdn_p0 sdn_p1 sdn_p2 sdn_p3
```

To use all available links to achieve increased throughput (active / active links), the bond mode needs to be changed to TCP load balancing, where the link used for a particular packet is selected based on L3 addresses (IPv4 and IPv6) and L4 ports:

```
ovs-vsctl set port bond0 bond_mode=balance-tcp
```

Note

Packets are approximately evenly spread over all ports on the bridge; the capacity of the links is not considered and is assumed equal for all ports.

To enable LACP on a bond, so that the availability of links is actively monitored (needs to be done on at least one of the two machines involved in the bond):

```
ovs-vsctl set port bond0 lacp=active
```

The interval at which LACP packets are transmitted can be configured as being fast (1 s) or slow (30 s) by using one of the following:

```
ovs-vsctl set port bond0 other_config:lacp-time=slow
ovs-vsctl set port bond0 other_config:lacp-time=fast
```

To show the bond configuration:

```
ovs-appctl bond/show bond0
```

When LACP has been enabled, OvS rebalances flows amongst the available slaves. The rebalance interval can be configured by:

```
ovs-vsctl set port bond0 other_config:bond-rebalance-interval=100
```

Note

OvS sends packets to the LOCAL port for each bond. This results in the flow rule including output to the LOCAL port. This cannot be accelerated by Agilio OvS. To prevent this from occurring, and to achieve acceleration, packets must **not** be sent to the LOCAL port, which can be achieved by:

```
ovs-ofctl -OOpenflow13 mod-port bondbr0 bondbr0 no-forward
```

Note

Link Aggregation in OvS is based on the NORMAL rule; links will not be aggregated when the bond 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 bond bridge to only have the NORMAL rule. This additional bridge can be connected to the bond bridge using a patch port:

```
ovs-vsctl add-port br0 bond_patch1 -- set Interface bond_patch1 \
  type=patch options:peer=bond_patch2
ovs-vsctl add-port bondbr0 bond_patch2 -- set Interface bond_patch2 \
  type=patch options:peer=bond_patch1
```

3.6 Virtual Function Configuration

The following sections describe different Virtual Function (VF) configurations available in the Agilio OvS software. For each of these configurations, the Bus/Device/Function (BDF) address will be required for each individual VF.

Each NFP PCIe unit has a different PCIe physical function on the PCIe bus and each such physical function can have a corresponding set of "slave" VFs. One can find a list of the BDF prefixes for all the VFs on the NFP PCIe units by performing:

```
lspci -d 19ee:6003
```

This should produce output containing multiple lines of the form:

```
XX:0Y.Z Ethernet controller: Netronome Systems, Inc. Device 6003
```

The VFs on any PCIe unit that are available on the system will be at BDF addresses of the form XX:0Y:Z with Y ranging from 8 to f and Z ranging from 0 to 7. For example, if the output shows `c4:08.3 Ethernet controller...` then VF 3 on NFP PCIe unit 0 will be at `c4:08.3`. The PCIe BDF address of a particular VF can most easily be found by running `ethtool -i` on the relevant representative netdev, for example:

```
ethtool -i sdn_v0.2 | grep bus-info
bus-info: NFP 0 VF0.2 0000:c4:08.2
```

One can use the `/opt/netronome/libexec/dpdk_nic_bind.py` command to bind specific drivers to VFs. The Agilio OvS software provides two different drivers which may be bound to VFs, the Netronome DPDK driver `nfp_uio` and the Netronome netdev driver `nfp_netvf`.

The user may use VFs 0 to 59 on each of the NFP's PCIe units. VFs 60 to 63 on each NFP PCIe unit are reserved for internal use. Three of these VFs (VFs 60, 61, and 62 on NFP PCIe unit 0) are bound to the `nfp_netvf` driver when the software is started. The resulting netdevs, named `sdn_ctl`, `sdn_pkt`, and `sdn_pri` respectively, are used for communication between the kernel and the NFP. **These netdevs should not be used or reconfigured in any way by the user. Ensure that third party applications (e.g. GNOME NetworkManager) do not try to manage these netdevs.**

3.6.1 Creating VF Network Devices

The Agilio OvS software VF may be used like any other netdev made available by the Linux kernel. In order to create such a netdev, one must bind the `nfp_netvf` driver to a particular VF via its BDF identifier. Refer to Section 3.6 on how to determine the BDF identifier for a particular VF.

Creating a VF netdev can be accomplished by executing the following command with the BDF identifier for the particular VF:

```
/opt/netronome/libexec/dpdk_nic_bind.py -b nfp_netvf c4:08.2
```

Standard Linux netdev naming rules, for example udev rules, will apply when the netdev is created. The name of the netdev can be determined by observing the output in the kernel ring buffer for the specific BDF identifier. For example the following output may be seen after binding the netdev driver to a VF:

```
[ 5947.941387] nfp_netvf 0000:c4:08.2 eth3: Netronome NFP-6xxx VF Netdev: ...
```

This indicates the netdev name is `eth3` for the VF with BDF identifier of `c4:08.2`.

3.6.2 Running a DPDK Application on NFP VFs

Running a DPDK application while binding it to Agilio OvS VFs is usually accomplished by passing command line arguments to the DPDK application. Some of these arguments go to the poll-mode driver (PMD) library used to interface with the VFs. To produce said arguments, one must first know the BDF identifiers that correspond to each VF. Refer to Section 3.6 on how to determine the BDF identifier for a particular VF.

The VFs are required to be bound to the Netronome DPDK driver `nfp_uio` before starting the DPDK application. This can be accomplished by executing the following command with the BDF identifier of the particular VF:

```
/opt/netronome/libexec/dpdk_nic_bind.py -b nfp_uio c4:08.2
```

Here is an example running the DPDK l2fwd application on three VFs to bump traffic inline.

```
# Build the application
cd /opt/netronome/srcpkg/dpdk-ns/examples/l2fwd
make RTE_SDK=/opt/netronome/srcpkg/dpdk-ns RTE_TARGET=x86_64-native-linuxapp-gcc

# Configure hugepages (see DPDK manuals)
mkdir -p /mnt/hugetlbfs
echo 1200 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
mount -thugetlbfs none /mnt/hugetlbfs

# Start the l2fwd application on VFs c4:08.1, c4:08.2 and c4:08.3
build/l2fwd -c f -n 4 -d /opt/netronome/lib/librte_pmd_nfp_net.so \
    --log-level=7 -w c4:08.1 -w c4:08.2 -w c4:08.3 -- -p 7
```

It should be noted that the number of hugepages will be allocated per NUMA-Node (two nodes will allocate twice the requested amount). Furthermore, at least 2 GiB of RAM should be left free for the kernel, the OvS datapath and Agilio OvS. Failing to adhere to this requirement could lead to the Linux Out-of-Memory (OOM) killer being invoked.

Most of the DPDK command line parameters are documented in DPDK itself. However, a few are worth noting in the example above:

1. `-d /opt/netronome/lib/librte_pmd_nfp_net.so` tells the application to use the NFP PMD. This is required.
2. `--log-level=7` significantly reduces the debug output in the system, crucial to achieving better performance.
3. Each `-w XX:YY.Z` adds another VF that the application will consider using, as a DPDK port numbered incrementally from 0. However the `-p N` specifies a bitmask of those ports that will actually be used by the application. In this example, `-p 7` indicates 0b111, indicating that all three ports (numbered 0, 1 and 2) are used.

See the DPDK documentation for more details.

Note

The l2fwd example application forwards traffic between pairs of adjacent VFs when transmitting. If the application opens an odd number of VFs, then the last VF will send traffic out the VF it came in on. Note also that the l2fwd application modifies the MAC addresses of packets that traverse it.

3.6.3 Virtualization Configuration

3.6.3.1 Host Setup for Virtualization

Intel Virtualization Technology (VT) and Intel Virtualization Technology for directed I/O (VT-d) must be enabled in the host's BIOS. Afterward, the kernel command line parameters must be set to enable IOMMU. This can be done as follow:

Edit grub config `/etc/default/grub`. Edit either GRUB_CMDLINE_LINUX or GRUB_CMDLINE_LINUX_DEFAULT and include the following parameters:

- `intel_iommu=on`
- `iommu=pt`

For example:

```
GRUB_CMDLINE_LINUX_DEFAULT="intel_iommu=on iommu=pt"
```

Update grub and reboot by entering:

- `update-grub2`
- `reboot`

After the system has rebooted, verify the kernel command line with the following command:

`cat /proc/cmdline`.

For example, the output would look similar to this:

```
BOOT_IMAGE=/boot/vmlinuz-3.13.0-46-generic \
root=UUID=baf2673a-84d1-4dc0-9f08-0b4e4d757de2 ro biosdevname=0 ghes.disable=1 \
console=ttyS0,115200n81 intel_iommu=on iommu=pt
```

Using virtualization with the Agilio OvS software requires `kvm` and `virt-manager` installed on the host system. These packages can be installed with the following commands on Ubuntu systems (similar packages should be installed on other Linux distributions using the relevant package management utility):

```
apt-get install kvm virt-manager
```

In order to allow `virt-manager` to memory map the Netronome libraries, the following changes must be made to the apparmor profile:

The following line

```
/opt/netronome/lib/** rm,
```

must be added to the `/etc/apparmor.d/usr.lib.libvirt.virt-aa-helper` file before

```
/{media,mnt,opt,srv}/** r,
```

Save the changes to the file and reload `apparmor` for the changes to take effect with the following commands:

```
service apparmor reload

service libvirt-bin restart
```

Start `virt-manager` and use the X11 GUI[1] to create a VM with the following recommended options:

- CPU: at least 4 CPU's (recommended for DPDK)
- CPU Configuration: Copy from host (this is required to compile DPDK)
- Memory (at least 2GB recommended)

Refer to the online documentation for `virt-manager` for more details to install and configure VMs.

3.6.3.2 Guest Setup for Virtualization

In order to setup virtualization on the guest system, all the Agilio OvS required packages listed in the GSG must be installed in the guest. Afterward, copy the Agilio OvS installation tarball to the guest and extract the tarball. Change to the extracted directory and enter `make vm_install`.

Installation is complete once the following message is displayed on screen:

```
 *** Agilio OvS Software 2.2 Virtual Machine Installation Complete!
     Build Number: 'build number'
```

The next step is to shutdown the VM and configure PCI passthrough.

[1]VMs can also be created with commandline tools by knowledgeable users.

3.6.3.3 PCI Passthrough using VFIO Setup

To configure PCI passthrough, Agilio OvS software should be running. To start Agilio OvS software enter the following command: `ovs-ctl start`

Once Agilio OvS software is running, the PCI addresses of the VFs can be determined from the corresponding representative netdevs in the BDF format. Refer to Section 3.6 to determine the BDF identifier for a particular VF. The selected BDF addresses must be replaced in the commands below.

Enter the following commands as root on the host system to install the VFIO module and set it to accept Netronome VFs as valid passthrough devices:

- `modprobe vfio-pci`
- `echo "19ee 6003" > /sys/bus/pci/drivers/vfio-pci/new_id`

To bind the VFs to the vfio-pci passthrough driver, execute the following command for each VF:

- `/opt/netronome/libexec/dpdk_nic_bind.py -b vfio-pci <PCI_BDF>`

(where <PCI_BDF> is a bus:device.function address, for example 05:08.1)

On the VM system, using `virt-manager`, click on "Add Hardware", choose PCI Host Device and pick the correct VFs. After starting the VM, the VFs should be visible as PCI devices. Either the `nfp_uio` (the Netronome DPDK module) or the `nfp_netvf` (the Netronome netdev module) can be bound to the devices individually. Refer to Section 3.6.1 and Section 3.6.2 covering netdev and DPDK configuration respectively for VFs. To determine the BDF addresses of all the VFs inside the VM the following command may be used:

```
lspci -d 19ee:6003
```

The following VM configuration has been observed to cause IOMMU mapping errors when rebooting the VM and should be avoided. If such a host/guest configuration is required, the recommended solution is to power off the VM and start it up again instead of rebooting the guest. This problem is not specific to Netronome hardware.

The IOMMU mapping errors have been observed on the following host systems:

- CPU Architecture: Ivy Bridge
- Linux Kernel: 3.13.0-46 and 3.13.0-49
- Linux Distributions: Ubuntu 14.04
- Kernel Parameters:
  - `intel_iommu=on`
  - `iommu=pt`
  - `default_hugepagesz=1G`
  - `hugepagesz=1G`
  - `hugepages=8`

The guest system must be a QEMU VM with the following parameters passed to QEMU:

- `-enable-kvm`

- `-device pci-assign`
- `-mem-path /mnt/huge`

The following errors are expected to be seen on the host system in the kernel ring buffer:

```
[ 1798.594046] dmar: DRHD: handling fault status reg 202
[ 1798.596735] dmar: DMAR:[DMA Read] Request device [05:08.1] fault addr 12800000
[ 1798.596735] DMAR:[fault reason 06] PTE Read access is not set
```

This may cause long latencies and possibly cause the host system to become unstable.

3.6.4 Checksum Offload Configuration

3.6.4.1 Linux Kernel Netdev (nfp_netvf driver)

The Agilio OvS checksum offload capabilities are enabled by default when a VF is bound to the `nfp_netvf` driver. Both RX and TX checksum offloading is supported.

`ethtool` can be used to view the status of RX and TX checksum offloading:

```
ethtool -k eth0 | grep checksum
rx-checksumming: on
tx-checksumming: on
        tx-checksum-ipv4: on
        tx-checksum-ip-generic: off [fixed]
        tx-checksum-ipv6: on
        tx-checksum-fcoe-crc: off [fixed]
        tx-checksum-sctp: off [fixed]
```

To disable RX checksum offloading:

```
ethtool -K eth0 rx-checksum off
Actual changes:
rx-checksumming: off
```

To disable TX checksum offloading:

```
ethtool -K eth0 tx-checksum-ipv4 off tx-checksum-ipv6 off
Actual changes:
tx-checksumming: off
        tx-checksum-ipv4: off
        tx-checksum-ipv6: off
```

In both cases, replacing "off" with "on" will enable the feature. The Agilio OvS software makes no distinction between IPv4 and IPv6 where TX checksum offloading is concerned. Thus if either is enabled, the feature will be enabled for both. If the user desires to disable the feature, both need to be disabled.

3.6.4.2 DPDK (nfp_uio driver)

Agilio OvS supports the same checksum offload features for DPDK as the Linux kernel network driver. Refer to the DPDK documentation on how to configure the offloads programmatically.

3.6.5 Quality of Service Configuration

3.6.5.1 VM rate limiter configuration

The rate limiters can be used to limit traffic to and/or from a VF(Virtual Function). Since a VF is used by a Virtual Machine (VM), this will effectively rate limit the traffic to and from the VM. Egress rate limiters limit traffic going from the virtual switch to a VF. Ingress rate limiters limit traffic coming from a VF to the virtual switch. The following commands can be used to configure VF rate limiters.

```
ovs-vsctl set interface <intf> ingress_policing_rate=<rate>
ovs-vsctl set interface <intf> ingress_policing_burst=<burst>

ovs-vsctl set interface <intf> egress_policing_rate=<rate>
ovs-vsctl set interface <intf> egress_policing_burst=<burst>
```

Where <intf> is sdn_v0.0 - sdn_v0.59, or sdn_v1.0 - snd_v1.59 depending on the amount of PCIE buses in the system. <rate> is the maximum allowed rate, in kilobits per second (Kbps) burst is the maximum burst size, in kilobits.

The <rate> parameter can be between 10 (10Kbps) and 40000000 (40Gbps).

Note that a too large burst size can cause the actual rate to be higher than the desired rate. For very low rate and small burst size configurations, the burst size is adjusted to allow at least one jumbo packet to be processed by the rate limiter.

Rate limiters is refreshed every 5 to 10ms. Measuring the rate limiter output rate over small time intervals (e.g. 5 to 10ms) will show inaccuracies. However, these will average out over longer intervals (1s or more).

3.6.5.2 VM rate limiter example:

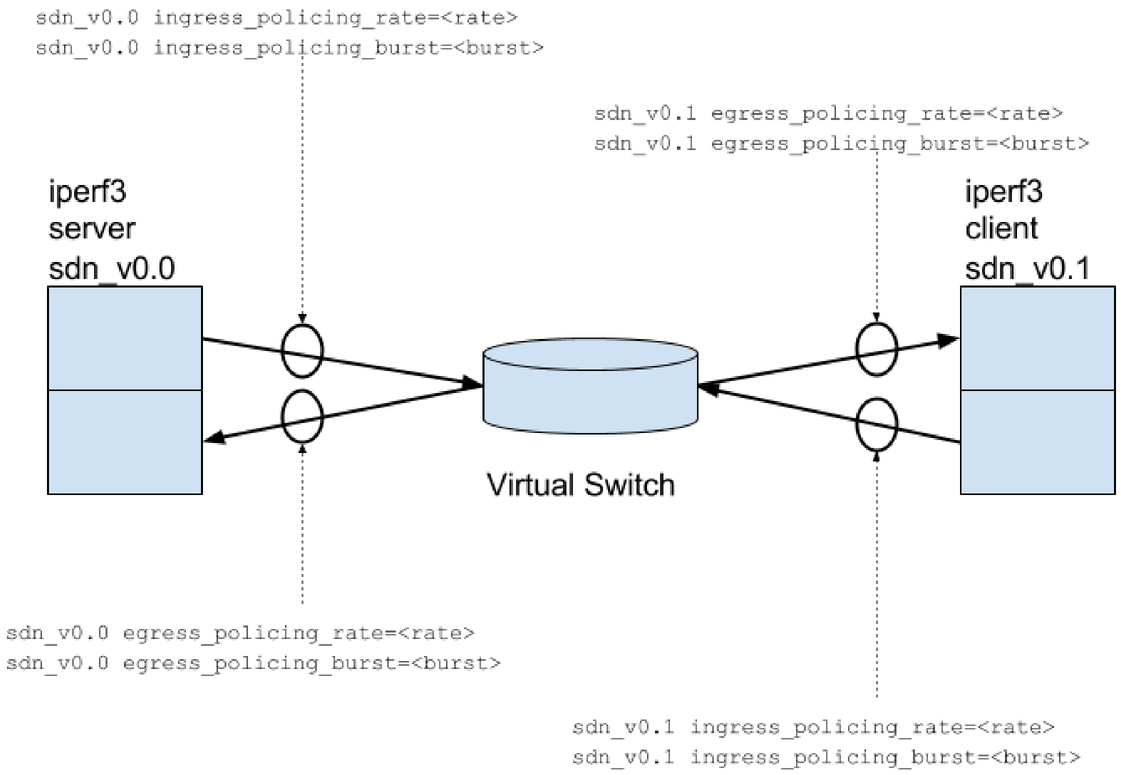
sdn_v0.0 ingress_policing_rate=<rate>
sdn_v0.0 ingress_policing_burst=<burst>
sdn_v0.1 egress_policing_rate=<rate>
sdn_v0.1 egress_policing_burst=<burst>
iperf3 server sdn_v0.0
iperf3 client sdn_v0.1
Virtual Switch
sdn_v0.0 egress_policing_rate=<rate>
sdn_v0.0 egress_policing_burst=<burst>
sdn_v0.1 ingress_policing_rate=<rate>
sdn_v0.1 ingress_policing_burst=<burst>

Figure 3.1. VM Rate limiter example setup

The image above shows the setup used to test the rate limiters using iperf3. The server runs on VF0 (sdn_v0.0) and the client runs on VF1 (sdn_v0.1). In this type of test the performance is measured by the amount of data the client sends to the server, i.e. the bottom arrow, going from right to left. To effectively rate limit traffic in this scenario, one only needs the sdn_v0.0 egress limiter or the sdn_v0.1 ingress rate limiter. If both are used, the resultant rate will be determined by the limiter with the lowest rate.

Server setup:

```
SDN_VF_0=sdn_v0.0
# This setup assumes sdn_v0.0 is already bound to nfp_netvf,
# and that the netdev name is eth0
if_0=eth0
# create bridge with 2 ports
ovs-vsctl --if-exists del-br $BR
ovs-vsctl add-br $BR
ovs-ofctl del-flows -OOpenflow13 br0
ovs-vsctl set bridge $BR protocols=OpenFlow13
#create ns1
ip netns add ns1
ovs-vsctl add-port br0 $SDN_VF_0 -- set Interface $SDN_VF_0 ofport_request=1
ip link set $if_0 netns ns1
ip netns exec ns1 ifconfig $if_0 12.0.0.1/24 mtu 9000 up
ip netns exec ns1 ifconfig
#add flows
```

```
ovs-ofctl add-flow -OOpenflow13 br0 in_port=1,actions=output:2
ovs-ofctl add-flow -Oopenflow13 br0 in_port=2,actions=output:1
ip netns exec ns1 iperf3 -s
```

Client setup:

```
SDN_VF_1=sdn_v0.1
# This setup assumes sdn_v0.1 is already bound to nfp_netvf,
# and that the netdev name is eth1
if_1=eth1
ip netns add ns2
ovs-vsctl add-port br0 $SDN_VF_1 -- set Interface $SDN_VF_1 ofport_request=2
ip link set $if_1 netns ns2
ip netns exec ns2 ifconfig $if_1 12.0.0.2/24 mtu 9000 up
echo "ip netns exec ns2 iperf3 -c 12.0.0.1 -i2 -O2 -t30"
```

Rate limiter setup:

```
ovs-vsctl set interface sdn_v0.1 ingress_policing_rate=2000 #2Mbit/sec rate
ovs-vsctl set interface sdn_v0.1 ingress_policing_burst=100000 #100KB burst
```

Result:

```
ip netns exec ns2 iperf3 -c 12.0.0.1 -i3 -O2 -t30
Connecting to host 12.0.0.1, port 5201
[  4] local 12.0.0.2 port 60101 connected to 12.0.0.1 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-3.00   sec   769 KBytes  2.10 Mbits/sec   56   26.2 KBytes
[  4]   3.00-6.00   sec   769 KBytes  2.10 Mbits/sec   56   26.2 KBytes
[  4]   6.00-9.00   sec   734 KBytes  2.00 Mbits/sec   56   26.2 KBytes
[  4]   9.00-12.00  sec   612 KBytes  1.67 Mbits/sec   53   26.2 KBytes
[  4]  12.00-15.00  sec   734 KBytes  2.00 Mbits/sec   55   26.2 KBytes
[  4]  15.00-18.00  sec   786 KBytes  2.15 Mbits/sec   56   26.2 KBytes
[  4]  18.00-21.00  sec   769 KBytes  2.10 Mbits/sec   56   26.2 KBytes
[  4]  21.00-24.00  sec   734 KBytes  2.00 Mbits/sec   57   26.2 KBytes
[  4]  24.00-27.00  sec   769 KBytes  2.10 Mbits/sec   56   26.2 KBytes
[  4]  27.00-30.00  sec   786 KBytes  2.15 Mbits/sec   55   26.2 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-30.00  sec  7.29 MBytes  2.04 Mbits/sec  556             sender
[  4]   0.00-30.00  sec  7.29 MBytes  2.04 Mbits/sec                  receiver

iperf Done.
```

Note the -O2 parameter when invoking the iperf3 client. This is to ignore the initial TCP slow start and the burst that follows due to the rate limiter allowing a 100KB burst.

3.6.5.3 VM QoS

The VM QoS feature allows the user to define up to 8 different traffic classes for use in egress scheduling. The class is determined by the VF or network port. Each physical egress network port can have relative bandwidths assigned to each of the 8 classes. The scheduling algorithm used is Deficit Weighted Round Robin (DWRR).

The DWRR scheduling algorithm works as follows: Bandwidth will be allocated to each class in a relative fashion. Each class will be programmed with a unique weight. Weights are assigned in a relative manner across the classes according to the desired allocation of intended bandwidth for each class. For example, in an oversubscribed application if it is desired that one class of service receives 100 times the bandwidth as compared to a different class, the weight for that class would be programmed to 100x the other class.

- DWRR Example, Sum of all weights = 402,000
- Class 0, large weight = 200,000 : Relative bandwidth = 49.75%
- Class 1, medium weight = 100,000 : Relative bandwidth = 24.88%
- Class 2, medium weight = 100,000 : Relative bandwidth = 24.88%
- Class 3, small port weight = 2,000 : Relative bandwidth = 0.50%

Note that if no traffic of a certain class is scheduled, the relative bandwidth of that class can be used by the other classes. E.g in the example above, if there is only traffic assigned to class 3, it can use 100% of the bandwidth. If at the same time, traffic assigned to classes 0, 1, and 2 appear, the traffic assigned class 3 will be dropped up until the point where it reaches its assigned relative bandwidth.

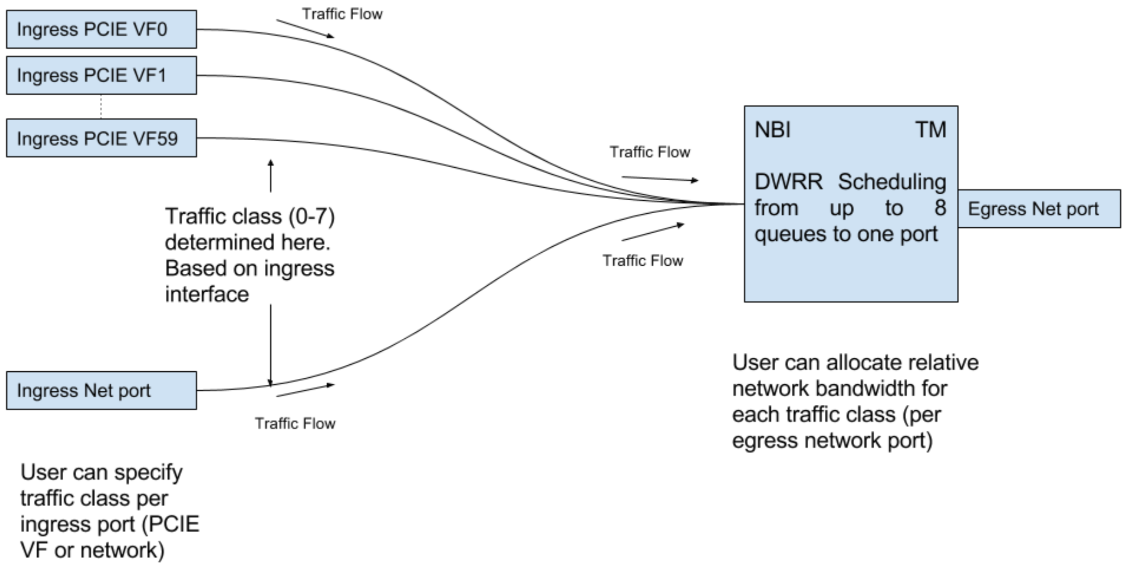
Ingress PCIE VF0
Ingress PCIE VF1
Ingress PCIE VF59
Traffic Flow
Traffic Flow
Traffic Flow
Traffic Flow
Traffic class (0-7) determined here. Based on ingress interface
NBI
TM
DWRR Scheduling from up to 8 queues to one port
Egress Net port
Ingress Net port
User can allocate relative network bandwidth for each traffic class (per egress network port)
User can specify traffic class per ingress port (PCIE VF or network)

Figure 3.2. VM QoS overview

3.6.5.4 VM QoS example

Clear QoS (per port or all):

```
ovs-vsctl clear port sdn_p0 qos
ovs-vsctl clear port sdn_p1 qos
ovs-vsctl --all destroy qos
```

Add new QoS class to port (sdn_p0 is assigned class 2, sdn_p1 is assigned class 0). Also applicable to VF's (sdn_vx.y)

```
ovs-vsctl set port sdn_p0 qos=@newqos -- \
--id=@newqos create qos type=nfp-dwrr-qos other-config:ingress-class=2
ovs-vsctl set port sdn_p1 qos=@newqos -- \
--id=@newqos create qos type=nfp-dwrr-qos other-config:ingress-class=0
```

Modify Ingress Traffic Class per port:

```
ovs-vsctl list port sdn_p0      # Observe _UUID for QoS entry
ovs-vsctl set qos 0e5544fb-d90a-4128-8a0b-431ab8ff0233 other_config:ingress-class=1
```

View all QoS entries:

```
ovs-vsctl list qos
```

Change the DWRR weight of a class at an egress port:

```
ovs-vsctl set port sdn_p1 qos=@newqos -- \
--id=@newqos create qos type=nfp-dwrr-qos other-config:ingress-class=2 \
queues:0=@q0 queues:1=@q1 queues:2=@q2 queues:3=@q3 \
queues:4=@q4 queues:5=@q5 queues:6=@q6 queues:7=@q7 -- \
--id=@q0 create Queue other-config:dwrr-relative-bandwidth=1500 -- \
--id=@q1 create Queue other-config:dwrr-relative-bandwidth=10000000 -- \
--id=@q2 create Queue other-config:dwrr-relative-bandwidth=5000000 -- \
--id=@q3 create Queue other-config:dwrr-relative-bandwidth=2000000 -- \
--id=@q4 create Queue other-config:dwrr-relative-bandwidth=40000 -- \
--id=@q5 create Queue other-config:dwrr-relative-bandwidth=5000 -- \
--id=@q6 create Queue other-config:dwrr-relative-bandwidth=4000 -- \
--id=@q7 create Queue other-config:dwrr-relative-bandwidth=1500
```

In the above example, classes 0 and 7 will be allocated 0.0088% of the relative bandwidth. In the case where a 10Gbps port is oversubscribed, classes 0 and 7 will each be allocated 880Kbps. Note that is is not possible to set relative bandwidth weight to absolute 0.

The minimum value of "dwrr-relative-bandwidth" is 1. However, the user should not set the minimum lower than the largest expected frame size. As a guideline, values smaller than the interface MTU should not be used. The maximum value is 16777215. Attempting to set a value of 0 will have no effect.

The ovs-vsctl command above is just an example to configure all 8 queues at once. Queues can be added one by one or in any other fashion. See below.

To modify only a single queue after it has been configured:

```
ovs-vsctl set queue <UUID> other_config:dwrr-relative-bandwidth=1000
```

UUID can be determined with 'ovs-vsctl list queue'

It is important to note that if a user disconnects the QoS class from the port without clearing the queues first (as shown above), the code cannot reset the queues to its default state and the last QoS configuration will remain active on the particular port.

To view QoS configuration, it is recommended to use this utility:

```
ovs-appctl qos/show sdn_p0
```

If an invalid configuration is attempted, the above ovs-appctl command will show what the actual configuration is. OVSDB will unfortunately show the invalid configuration and users may think the configuration was successful.

Note that by default all ingress ports and VF's are set to class 0. On the egress network ports, class 0 has a DWRR weight of 200000 and classes 1-7 have weights of 1. If the user decides to give precedence to VM traffic over network-network traffic, he/she is responsible to expressly configure it as such.

3.7 VirtIO Relay

The optional VirtIO relay daemon (`virtiorelayd`) can relay packets between SR-IOV VFs on the host, and VirtIO devices in a QEMU guest VM. The benefit of this is that any guest OS supporting VirtIO can be used, at the expense of losing the native SR-IOV performance advantage. In a nutshell, VirtIO relay monitors OVSDB to discover which SR-IOV VF ports to relay, uses DPDK to interface with such VFs, and uses vhost-user unix sockets to exchange VirtIO information with QEMU virtual machines. VFs can currently only be relayed 1:1 with VirtIO instances going to VMs.[2]

To enable this daemon, add the following to `/etc/netronome.conf`:

```
SDN_VIRTIORELAY_ENABLE=y
```

This must be done prior to starting Agilio OVS with the `ovs-ctl start` command. The `ovs-ctl status` command will also show the status of the VirtIO relay. Further configuration details are supplied below.

3.7.1 Requirements

Some external requirements must be met in order to use VirtIO relay successfully:

- QEMU version 2.5 (or newer) must be used for the virtual machine hypervisor. The older QEMU 2.3 and 2.4 do work with `virtiorelayd`, though there are bugs[3], less optimised performance and missing features[4]. Some specific QEMU settings are required, detailed in a subsection below.
- libvirt 1.2.6 or newer (if using libvirt to manage VMs, manually scripted QEMU commandline VMs don't require libvirt)
- 2M hugepages must be configured in Linux, a corresponding hugetlbfs mountpoint must exist, and at least 320 hugepages must be free for use by VirtIO relay.

3.7.2 libvirt and apparmor

Libvirt's apparmor permissions might need to be modified to allow read/write access to the hugepages directory and library files for QEMU:

```
# in /etc/apparmor.d/abstractions/libvirt-qemu
# for latest QEMU
/usr/lib/x86_64-linux-gnu/qemu/* rmix,
# for access to hugepages
owner "/mnt/huge/libvirt/qemu/**" rw,
owner "/mnt/huge-1G/libvirt/qemu/**" rw,
```

Be sure to substitute the hugetlbfs mountpoints that you use into the above.

[2]Future versions of VirtIO relay may make it possible to combine many VFs into fewer VirtIO pipes, or few VFs into many VirtIO instances.
[3]In particular, shared memory hugepages may be leaked when restarting VMs.
[4]For example, QEMU 2.5 is required to use VirtIO 1.0 successfully.

3.7.3 hugepages

VirtIO relay requires 2M hugepages and QEMU/KVM can give better performance with 1G hugepages. To set up the system for use with QEMU and VirtIO relay, the following should be added to the Linux kernel command line parameters:

```
hugepagesz=2M hugepages=768 default_hugepagesz=1G hugepagesz=1G hugepages=8
```

The following could be done after each boot:

```
#Reserve at least 320 * 2M for virtio-relay:
echo 768 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
#Reserve 8G for application hugepages (modify this as needed for VMs):
echo 8 > /sys/kernel/mm/hugepages/hugepages-1048576kB/nr_hugepages
```

Note that reserving hugepages after boot may fail if not enough contiguous free memory is available, and it is therefore recommended to reserve them at boot time with Linux kernel command line parameters. This is especially true for 1G hugepages.

hugetlbfs needs to be mounted on the filesystem to allow applications to create and allocate handles to the mapped memory. The following lines mount the two types of hugepages on /mnt/huge (2M) and /mnt/huge-1G (1G):

```
grep hugetlbfs /proc/mounts | grep -q "pagesize=2M" || \
( mkdir -p /mnt/huge && mount nodev -t hugetlbfs -o rw,pagesize=2M /mnt/huge/ )
grep hugetlbfs /proc/mounts | grep -q "pagesize=1G" || \
( mkdir -p /mnt/huge-1G && mount nodev -t hugetlbfs -o rw,pagesize=1G /mnt/huge-1G/ )
```

Finally, if libvirt is to be used, libvirt requires a special directory inside the hugepages mounts with the correct permissions in order to create the necessary per-VM handles:

```
mkdir /mnt/huge-1G/libvirt
mkdir /mnt/huge/libvirt
chown libvirt-qemu:kvm -R /mnt/huge-1G/libvirt
chown libvirt-qemu:kvm -R /mnt/huge/libvirt
```

On CentOS the `libvirt-qemu` username above needs to be substituted with the `qemu` username.

Note

After these mounts have been prepared, the libvirt daemon will probably need to be restarted.

It is also possible (and simpler) to have the VMs use 2M hugepages as well instead of 1G hugepages, then one does not need to manually reserve 1G hugepages at boot time. There is a performance benefit to using 1G hugepages for the VMs however, especially if they use a few gigabytes of RAM.

3.7.4 VirtIO relay configuration

To enable this daemon, add the following to `/etc/netronome.conf`:

```
SDN_VIRTIORELAY_ENABLE=y
```

The daemon can be further configured by command line parameters, which can be specified in the following variable:

```
SDN_VIRTIORELAY_PARAM="--cpus=1,2 --vhost-username=libvirt-qemu\
--vhost-groupname=kvm --huge-dir=/mnt/huge"
```

By default, when this variable is not specified, VirtIO relay will use CPUs 1 and 2, the vhost username and group will be set according to the Linux distribution in use and the hugepage directory will be autodetected from the mount points. The available command line options are:

```
USAGE: virtiorelayd --cpus=<val> [--nodaemon] [--loglevel=<val>] [--pid-path=<val>] \
[--huge-dir=<val>] [--ovsdb-sock=<val>] [--vhost-username[=<val>]] \
[--vhost-groupname[=<val>]] [--vhost-path=<val>] [--vhost-socket=<val>]

VirtIO Relay daemon: forward packets between SR-IOV VFs (serviced by DPDK) and
VirtIO network backend.

Options:
  -C --cpus <val>                CPUs to use for worker threads (either comma
                                 separated integers or hex bitmap starting with
                                 0x)
  -n --nodaemon                  Don't daemonize (default: run as daemon)
  -l --loglevel <val>            Set log threshold 0-7 (least to most verbose)
                                 (default: 6)
  -p --pid-path <val>            PID file (virtiorelayd.pid) will be written to
                                 this directory (default: /var/run)
  -H --huge-dir <val>            The mount path to the hugepages (default:
                                 /mnt/huge)
  -O --ovsdb-sock <val>          OVSDB unix domain socket file (default:
                                 /usr/local/var/run/openvswitch/db.sock)
  -u --vhost-username [<val>]    vhost-user unix socket ownership username, omit
                                 value to inherit process username (default:
                                 libvirt-qemu)
  -g --vhost-groupname [<val>]   vhost-user unix socket ownership groupname, omit
                                 value to inherit process groupname (default:
                                 kvm)
  -V --vhost-path <val>          vhost-user unix socket directory path (default:
                                 /tmp)
  -S --vhost-socket <val>        vhost-user unix socket file name, must contain
                                 exactly one %u to denote VirtIO ID (default:
                                 virtiorelay%u.sock)
```

3.7.5 Adding VF ports to VirtIO relay

SR-IOV VF ports are added to an Agilio OVS bridge using the representative netdevs and the `ovs-vsctl` command (see Section 3.3.2). Similarly, to add a VF to the VirtIO relay, the `ovs-vsctl` command can be used with a special `external_ids` value containing an indication to use the relay. The bridge name `br-virtio` in this example is arbitrary, any bridge name may be used:

```
ovs-vsctl add-port br-virtio sdn_v0.42 -- set interface sdn_v0.42 \
external_ids:virtio_relay=1
```

The `ovs-vsctl` command manipulates the OVSDB, which is monitored for changes by VirtIO relay. The preceding command can be interpreted as follows: the VF represented by `sdn_v0.42` will be configured to be relayed to the VirtIO ID 42. This ID is present in the vhost-user unix socket path, which is passed to the QEMU VM (see next section), by default this will be `/tmp/virtiorelay42.sock`. The ID can be any value from 0 to 59, and is determined from the VF used. Note that the ports in the OVSDB remain configured across OVS restarts, and when `virtiorelayd` starts it will find the initial list of ports with associated VirtIO relay indications and recreate the necessary associations.

Changing an interface with no VirtIO relay indication to one with a VirtIO relay indication, or changing one with a VirtIO relay indication to one without a VirtIO relay indication also works. e.g.

```
# add to OVS bridge without VirtIO relay (ignored by VirtIO relay)
ovs-vsctl add-port br-virtio sdn_v0.2
# add VirtIO relay (detected by VirtIO relay)
ovs-vsctl set interface sdn_v0.2 external_ids:virtio_relay=1
# remove VirtIO relay (detected by VirtIO relay and removed from relay,
#  but remains on OVS bridge)
ovs-vsctl remove interface sdn_v0.2 external_ids virtio_relay
```

The externals_ids of a particular interface can be viewed with ovs-vsctl as follows:

```
ovs-vsctl list interface sdn_v0.1|grep external_ids
```

A list of all the interfaces with external_ids can be queried from OVSDB:

```
ovsdb-client --pretty -f list dump Interface name external_ids | \
grep -A2 -E "external_ids.*: {.+}"
```

This will result in output similar to:

```
external_ids        : {virtio_relay="1"}
name                : "sdn_v0.1"

external_ids        : {virtio_relay="1"}
name                : "sdn_v0.2"
```

3.7.6 VirtIO relay CPU affinity

Currently VirtIO relay does not provide any control w.r.t. which CPU is used for a particular VF/virtio relay pair, and newly added relay pairs are simply assigned to the least busy CPU in the list of CPUs provided in the configuration. It is also not possible to utilise more than one CPU for a particular relay pair.[5]

3.7.7 Running virtual machines

QEMU virtual machines can be run manually on the command line, or by using libvirt to manage them. To use QEMU manually with the vhost-user backed VirtIO which the VirtIO relay provides, the following example can be used:

[5] In future, the relay may provide more control over which CPU resources are assigned to which relay pairs, and make provision for using two different CPUs for a particular relay pair (splitting NFP-to-VM and VM-to-NFP traffic).

```
-object memory-backend-file,id=mem,size=3584M,mem-path=/mnt/huge-1G,share=on,prealloc=on \
-numa node,memdev=mem -mem-prealloc \
-chardev socket,id=chr0,path=/tmp/virtiorelay1.sock \
-netdev type=vhost-user,id=guest3,chardev=chr0,vhostforce \
-device virtio-net-pci,netdev=guest3,csum=off,gso=off,guest_tso4=off,guest_tso6=off,\
guest_ecn=off,mac=00:03:02:03:04:01
```

It is important for the VM memory to be marked as shareable (`share=on`) and preallocated (`prealloc=on` and `-mem-prealloc`), the mem-path must also be correctly specified to the hugepage mount point used on the system. The path of the socket must be set to the correct VirtIO relay vhost-user instance, and the MAC address may be configured as needed.

Virtual machines may also be managed using libvirt, and this requires some specific XML snippets in the libvirt VM domain specification file:

```
    <memoryBacking>
      <hugepages>
        <page size='1048576' unit='KiB' nodeset='0'/>
      </hugepages>
    </memoryBacking>

    <cpu mode='custom' match='exact'>
      <model fallback='allow'>SandyBridge</model>
      <feature policy='require' name='ssse3'/>
      <numa>
        <cell id='0' cpus='0-1' memory='3670016' unit='KiB' memAccess='shared'/>
      </numa>
    </cpu>
```

If only 2M hugepages are in use on the system, the domain can be configured with the following page size:

```
<page size='2' unit='MiB' nodeset='0'/>
```

Note, the emulated CPU requires SSSE3 instructions for DPDK support.

The following snippet illustrates how to add a vhost-user interface to the domain:

```
    <devices>
      <interface type='vhostuser'>
        <source type='unix' path='/tmp/virtiorelayRELAYID.sock' mode='client'/>
        <model type='virtio'/>
        <alias name='net1'/>
        <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
      </interface>
    </devices>
```

Note: when starting the domain, make sure that the permissions are correctly set on the relay vhost-user socket, as well as adding the required permissions to the apparmor profile. The `--vhost-username` and `--vhost-groupname` command line parameters to the VirtIO relay can also be used to set the permissions on the vhost-user sockets, these default to user:group of libvirt-qemu:kvm on Ubuntu, or qemu:kvm for CentOS.

It is recommended to pin the virtual CPUs to dedicated host CPUs, for example:

```
  <cputune>
    <vcpupin vcpu='0' cpuset='4'/>
    <vcpupin vcpu='1' cpuset='5'/>
  </cputune>
```

This can also be set at runtime with the `virsh vcpupin` command.

3.7.8 Running DPDK on VirtIO inside virtual machines

Instead of using a VirtIO netdev, DPDK can be used with VirtIO inside the VM with a script similar to:

```
mkdir -p /mnt/huge
echo 256 > /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
grep -q hugetlbfs /proc/mounts || mount -t hugetlbfs nodev /mnt/huge
modprobe uio
modprobe uio_pci_generic
/opt/netronome/libexec/dpdk_nic_bind.py -b uio_pci_generic 00:06.0
# Run a DPDK app with relevant parameters
./trafgen -c 0x3 -n 1 --proc-type auto --socket-mem 512 --file-prefix sink -w 0000:00:06.0 \
-- -b -p 0x1 -w 1a:46:0b:ca:bc:01 -x 1a:46:0b:ca:bc:02 -i 64 -y 64 -j 64 -t 32 -Q 900 -z 64
```

A handy way to test is to use VirtIO PMD inside the VM with trafgen sample app in benchmark mode, and set up an OVS flow to echo the VF traffic on the NFP using an OpenFlow in_port action, e.g.

```
ovs-vsctl add-br br-VF0
ovs-vsctl add-port br-VF0 sdn_v0.0 -- set interface sdn_v0.0 external_ids:virtio_relay=1
ovs-ofctl del-flows br-VF0
ovs-ofctl add-flow br-VF0 in_port=1,actions=in_port
```

3.7.9 Using VirtIO 1.0

To enable VirtIO 1.0, the backend virtual PCI device provided by QEMU needs to be enabled. Using QEMU 2.5, you need to supply an extra cmdline parameter to prevent VirtIO 1.0 support from being disabled (it is disabled by default, since there are apparently still known issues with performance, stability and live migration):

```
-global virtio-pci.disable_modern=off
```

This can be done in a libvirt domain by ensuring the domain spec starts with something like:

```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
```

and just prior to the closing `</domain>` tag adding the following:

```
<qemu:commandline>
  <qemu:arg value='-global'/>
  <qemu:arg value='virtio-pci.disable-modern=off'/>
</qemu:commandline>
```

In addition to this, the vhost or vhost-user connected to the device in QEMU must support VirtIO 1.0. The vhost-user interface which VirtIO relay supplies does support this, but if the host is running a Linux kernel older than 4.0, you likely won't have vhost-net (kernel) support for any network interfaces in your QEMU VM which are not connected to VirtIO relay, for example if you have a bridged management network interface. Libvirt will by default use vhost-net for that, you can disable vhost-net by adding `<driver name='qemu'/>` to the relevant bridge interface as follows:

```
<interface type='bridge'>
  ...
  <model type='virtio'/>
  <driver name='qemu'/>
  ...
</interface>
```

To use VirtIO 1.0 with DPDK inside a VM, you will need to use DPDK 16.04. To use a VirtIO 1.0 netdev in the VM, the VM must be running Linux kernel version 4.0 or newer.

3.7.10 Inspecting traffic with tcpdump

To inspect traffic going to/from the relayed VFs, the port mirroring feature detailed in Section 3.2.6 can be used to mirror packets on a VF to another VF running a netdev. For example, if you have a virtiorelay using VF0 and wish to inspect the traffic going to and from the VM, you could do the following:

```
ovs-ctl configure mirror rx from sdn_v0.0 to sdn_v0.40
ovs-ctl configure mirror tx to sdn_v0.0 to sdn_v0.40
ethtool -i sdn_v0.40   (note PCI ID)
/opt/netronome/libexec/dpdk_nic_bind.py -b nfp_netvf <PCI ID>
dmesg|tail   (note netdev name, e.g. eth4)
tcpdump -i <netdev>
```

3.7.11 Debug counters

Some debug counters (showing packet counts) from the running VirtIO relay can be retrieved using the `virtio_relay_stats` command, e.g.:

```
# /opt/netronome/bin/virtio_relay_stats
Netronome VirtIO relay
worker1.virtio_rx=5026612254
worker1.dpdk_tx=5026612254
worker1.dpdk_drop=0
worker1.dpdk_full=0
worker1.dpdk_rx=5026807160
worker1.virtio_tx=5026807160
worker1.virtio_drop=0
worker1.virtio_full=4189877
worker2.virtio_rx=5027206272
worker2.dpdk_tx=5027206272
worker2.dpdk_drop=0
worker2.dpdk_full=0
worker2.dpdk_rx=5026573624
worker2.virtio_tx=5026573624
worker2.virtio_drop=0
worker2.virtio_full=209216
```

3.8 Health Monitoring

The health of the Agilio OvS software and the platform hardware can be checked by invoking the following command:

```
/opt/netronome/bin/ovs-ctl status
```

The output of this command will indicate the current status (`PASS/FAIL/WARN/SKIPPED`) of each hardware / software component as well as a consolidated result which reflects the health of the system as a whole. The consolidated result is only `PASS` if none of the respective system components is in a `FAIL` state. A syslog message is generated for each of the failures or warnings that are detected at the component level.

The health monitoring utility could potentially skip certain tests when the system is in a state where that test cannot be performed. For example, the Flow Processing Core test will be skipped when the firmware is not loaded. Similarly, many tests will be skipped while the Agilio OvS software is being started or stopped.

The utility can also log warnings for some tests. This will happen when a system component is not in a failed state, but there is a condition which the user should be made aware of.

Note

It is recommended to not run the health monitoring utility while the system is being started or stopped.

The following syslog failure messages could potentially be generated when invoking the Agilio OvS health checking utility:

- *SDN Failure=(Userspace Process Not Running: ovsdb-server) affected_component=(sw.host) action=(restart.software)*
- *SDN Failure=(Userspace Process Not Running: ovs-vswitchd) affected_component=(sw.host) action=(restart.software)*
- *SDN Failure=(Userspace Process Not Running: virtiorelayd) affected_component=(sw.host) action=(restart.software)*
- *SDN Failure=(Kernel Module Not Loaded: nfp) affected_component=(sw.driver) action=(restart.drivers)*
- *SDN Failure=(Kernel Module Not Loaded: nfp_cmsg) affected_component=(sw.driver) action=(restart.drivers)*
- *SDN Failure=(Kernel Module Not Loaded: nfp_fallback) affected_component=(sw.driver) action=(restart.drivers)*
- *SDN Failure=(Kernel Module Not Loaded: nfp_offloads) affected_component=(sw.driver) action=(restart.drivers)*
- *SDN Failure=(Kernel Module Not Loaded: openvswitch) affected_component=(sw.driver) action=(restart.drivers)*
- *SDN Failure=(Firmware Not Loaded) affected_component=(sw.nfp) action=(restart.nfp)*
- *SDN Failure=(FPC Unresponsive) affected_component=(sw.nfp) action=(restart.nfp)*
- *SDN Failure=(NFP Control Channel Unresponsive) affected_component=(sw.nfp) action=(restart.nfp)*
- *SDN Failure=(Ingress NBI Backed Up) affected_component=(sw.nfp) action=(restart.nfp)*
- *SDN Failure=(NFP Not Detected) affected_component=(hw.nfp) action=(replace.nfp)*
- *SDN Failure=(ECC Error Detected) affected_component=(hw.nfp) action=(replace.nfp)*
- *SDN Failure=(Parity Error Detected) affected_component=(hw.nfp) action=(replace.nfp)*

- *SDN Failure=([warning] ERR47 Workaround not detected) affected_component=(sw.host) action=(patch.os)*
- *SDN Failure=([warning] NFP Configurator outdated) affected_component=(hw.nfp) action=(update.nfp)*

All syslog messages are generated with the `ns_sdn_health` tag, `local0` facility, and `error` or `warning` level.

The health utility is also provided in the form of a shared library which can be embedded into an user application. Refer to the Agilio OvS PRM for more details on the health utility library usage.

3.9 PXE Booting

The Intelligent Server Adapters along with the Agilio OvS software supports PXE booting a server via the physical Intelligent Server Adapter ports. However the PXE booting feature itself is independent of the Agilio OvS software as this occurs during the boot process of the server.

Only 64-bit UEFI boot mode is supported to PXE boot a server via the Intelligent Server Adapter ports. The server must be configured into UEFI boot mode, and the appropriate ports enabled in the BIOS. Refer to the server's BIOS documentation on how to configure this feature.

All the Intelligent Server Adapter ports are available to PXE boot from including support for breakout cables. By default only port 0 will be enabled for booting after the Agilio OvS installation. The ports available for booting can be observed by executing the following command:

```
# /opt/netronome/bin/nfp-hwinfo | grep eth.*boot
eth0.boot=1
eth1.boot=1
eth2.boot=0
eth3.boot=0
eth4.boot=0
```

To enable or disable any of the other ports for booting, execute the following command:

```
# /opt/netronome/bin/nfp-media --set-hwinfo ethN.boot=1
```

where `N = ethernet instance` as shown in the ‘nfp-hwinfo’ output above. The system must be rebooted for the changes to take effect.

If multiple bootable ports are available, the system will try to PXE boot on the first port with an active link starting from the lowest port number. If no bootable ports are configured, the PXE boot firmware will default to using port 0 for PXE booting.

3.10 Connection Tracking (Conntrack)

Conntrack is an extension to Agilio OvS that enables the stateful tracking of flows on the NFP datapath. This allows the integration of security group, access control list and firewall components directly onto the datapath. To use the Conntrack functionality described in this section it must be selected at install time as described in the Getting Started Guide.

Note
The Conntrack version of the software is in an experimental state which implies that this is not a production version yet. Customers can evaluate this software for deployment planning purposes, however bugs are expected at this stage. Please stay in close synchronization with Netronome support.

Note
Selecting to use Conntrack at install time will allocate more of the NFP resources to the state tracking. For this reason there will be a performance decrease when running in this mode.

3.10.1 Additional Match Fields

In addition to those match fields in Agilio OvS, several new fields are available in Conntrack mode. Firstly, the following connection states can be matched on – these are flags that can either be checked, unchecked or 'don't care':

- Tracked (trk) - the packet has been tracked by Conntrack
- New (new) - the first packet of a flow
- Established (est) - the packet is a member of an established flow
- Related (rel) - the packets flow is related to another flow (see helper section)
- Reply (rpl) - the packet is in the opposite direction to that which initialised the flow
- Invalid (inv) - Conntrack finds the packet invalid

As well as states, the following fields can be matched on:

- ct_zone (16-bit) - separates Conntrack lookups into zones so different networks can be treated separately
- ct_mark (32-bit) - metadata than can be set when a packet is sent to Conntrack
- ct_label (128-bit) - metadata than can be set when a packet is sent to Conntrack

3.10.2 Additional Action Fields

Using a Conntrack action 'ct(...)' sends the packet to the state tracking block. Once the state is determined it will be added to metadata and attached to the packet. If the 'recirculation' (table=n) action is also set then the packet will be passed back through the datapath. After this recirculation, the packet will be able to match rules featuring Conntrack match fields as featured above.

The following Conntrack options can be included as part of the ct(..) action:

- exec(set_field:n->ct_mark) - set the ct_mark to n - matches on recirculation
- exec(set_field:n->ct_label) - set the ct_label to n - matches on recirculation

- commit - add the entry to the Conntrack table
- table=n - recirculate the packet to table n after it passes through Conntrack
- zone=n - set ct_zone to n - this is used a tuple in the connection state lookup
- alg=(ftp|sip) - send the packet to a helper function - see next section

3.10.3 Helper Functionality

Conntrack includes the tracking of 'expected' flows. These are flows that have not yet been seen but are anticipated to arrive. An example of such a case is in a FTP Passive connection. Here an FTP control channel is set up and established over well known port 21. This control channel can then be used to negotiate further data channels over potentially random port numbers. Conntrack can examine the negotiation phase of the control channel to determine the ports that are agreed on for the data channel. It then 'expects' a flow to appear between these IP addresses and over the negotiated port. When a packet arrives that matches one of these expected flows, its state is marked as 'related' meaning that the packet's flow is related to, or set up by, an already established and accepted flow. Helper functionality is currently available on both FTP and SIP protocols. Packets can be monitored for port negotiation by using the 'alg=FTP' or 'alg=SIP' action.

3.10.4 Open vSwitch Configuration of Conntrack

As with other Agilio OvS 2.2 actions, Conntrack is also compatible with the ovs-ofctl app. The following show examples of how this can be used:

```
ovs-ofctl add-flow -O Openflow13 br0 "ct_state=-trk,tcp,action=ct(table=0)"

ovs-ofctl add-flow -O Openflow13 br0 "ct_state=+trk+new-est,tcp,action=2"

ovs-ofctl add-flow -O Openflow13 br0 "ct_state=+trk-new+est-rpl,tcp,action=2"

ovs-ofctl add-flow -O Openflow13 br0 "ct_state=+trk-new+est+rpl,tcp,action=2"
```

The first rule above matches any packet that comes into the system. It sends this packet to the Conntrack block and tells it to recirculate back to table 0. On recirculation the packet can then match any of the last 3 rules based on the state that is determined from Conntrack.

```
ovs-ofctl -O Openflow13 add-flow br0 "ct_state=-
trk,action=ct(commit,table=0,zone=123,exec(set_field:0xdead->ct_label,
set_field:0xbeef->ct_mark)"

ovs-ofctl -O Openflow13 add-flow br0
"ct_mark=0xbeef,ct_zone=123,ct_label=0xdead,ct_state=+trk,action=0"
```

Here, the top rule sends to Conntrack setting the ct_state, ct_mark, and ct_zone before recirculating. The second rule should match any recirculating packet as it matches on the same mark, label and zone that have been set by the first rule.

```
ovs-ofctl -O Openflow13 add-flow br0 "tcp,ct_state=-
trk,action=ct(alg=ftp,table=0,commit)"

ovs-ofctl -O Openflow13 add-flow br0 "ct_state=+trk-rel,action=drop"

ovs-ofctl -O Openflow13 add-flow br0 "ct_state=+trk+rel,action=2"
```

In this final example, the first rule sends all matching packets to Conntrack and has them examined for FTP data channel negotiation, then recirculated. The final 2 rules check for any new data channels by looking for the inclusion or emission of a 'related' flag in the packet state.

4. Technical Support

To obtain additional information, or to provide feedback, please email `<support@netronome.com>` or contact the nearest **Netronome** technical support representative.

4.1 Related Documents

Descriptive Name	Description
Netronome Network Flow Processor: Agilio OvS 2.2 Getting Started Guide	A guide to new users of Netronome's Agilio OvS software.
Netronome Network Flow Processor: Agilio OvS 2.2 Programmer's Reference Manual	A reference for programming and system design using the Agilio OvS software.
Intelligent Server Adapters: Hardware User Manual	Contains summary information on the Netronome Intelligent Server Adpater (ISA) PCIe card including card physical descriptions.
Netronome Network Flow Processor: Datasheet	Contains summary information on the Netronome Network Flow Processor NFP including a functional description, signal descriptions, electrical specifications, and mechanical specifications.
Netronome Network Flow Processor: Databook	Contains detailed reference information on the Netronome Network Flow Processor NFP.
Netronome Network Flow Processor: Development Tools User's Guide	Describes Programmer Studio and the development tools that can be accessed through Programmer Studio.
Netronome Network Flow Processor: Network Flow Assembler System User's Guide	Describes the syntax of the NFP's assembly language, supplies assembler usage information, and lists assembler warnings and errors.
Netronome Network Flow Processor: Microengine Programmer's Reference Manual	A reference for microcode programming of the Netronome Network Flow Processor NFP.
Netronome Network Flow Processor: Network Flow C Compiler User's Guide	Presents information, language structures and extensions to the language specific to the Netronome Network Flow C Compiler for Netronome NFP.
Netronome Network Flow Compiler LibC: Reference Manual	Specifies the subset and the extensions to the language that support the unique features of the Netronome Network Flow Processor NFP product line.
Open vSwitch Software Documentation	Agilio OvS software offers acceleration of Open vSwitch software. Refer to http://openvswitch.org/ for more details on Open vSwitch.
OpenFlow Specification	Open vSwitch (which is accelerated by Agilio OvS software) is an OpenFlow switch implementation. Refer to https://www.opennetworking.org/sdn-

Descriptive Name	Description
	resources/openflow for more details on this specification.
Data Plane Development Kit Documentation	DPDK related documentation is available at http://dpdk.org.