Open vSwitch

Navigation

Open vSwitch with DPDK

This document describes how to build and install Open vSwitch using a DPDK datapath. Open vSwitch can use the DPDK library to operate entirely in userspace.

Important

The **releases FAQ** lists support for the required versions of DPDK for each version of Open vSwitch. If building OVS and DPDK outside of the master build tree users should consult this list first.

Build requirements

In addition to the requirements described in **Open vSwitch on Linux, FreeBSD and NetBSD**, building Open vSwitch with DPDK will require the following:

- DPDK 18.11
- A **DPDK supported NIC**

  Only required when physical ports are in use

- A suitable kernel

  On Linux Distros running kernel version >= 3.0, only *IOMMU* needs to enabled via the grub cmdline, assuming you are using **VFIO**. For older kernels, ensure the kernel is built with `UIO`, `HUGETLBFS`, `PROC_PAGE_MONITOR`, `HPET`, `HPET_MMAP` support. If these are not present, it will be necessary to upgrade your kernel or build a custom kernel with these flags enabled.

Detailed system requirements can be found at **DPDK requirements**.

Installing

Install DPDK

1. Download the **DPDK sources**, extract the file and set `DPDK_DIR`:

```
$ cd /usr/src/
$ wget http://fast.dpdk.org/rel/dpdk-18.11.tar.xz
$ tar xf dpdk-18.11.tar.xz
$ export DPDK_DIR=/usr/src/dpdk-18.11
$ cd $DPDK_DIR
```

2. (Optional) Configure DPDK as a shared library

   DPDK can be built as either a static library or a shared library. By default, it is configured for the former. If you wish to use the latter, set `CONFIG_RTE_BUILD_SHARED_LIB=y` in `$DPDK_DIR/config/common_base`.

   Note

   Minor performance loss is expected when using OVS with a shared DPDK library compared to a static DPDK library.

3. Configure and install DPDK

   Build and install the DPDK library:

```
$ export DPDK_TARGET=x86_64-native-linuxapp-gcc
$ export DPDK_BUILD=$DPDK_DIR/$DPDK_TARGET
$ make install T=$DPDK_TARGET DESTDIR=install
```

4. (Optional) Export the DPDK shared library location

   If DPDK was built as a shared library, export the path to this library for use when building OVS:

```
$ export LD_LIBRARY_PATH=$DPDK_DIR/x86_64-native-linu
```

Install OVS

OVS can be installed using different methods. For OVS to use DPDK datapath, it has to be configured with DPDK support (`--with-dpdk`).

Note

This section focuses on generic recipe that suits most cases. For distribution specific instructions, refer to one of the more relevant guides.

1. Ensure the standard OVS requirements, described in **Build Requirements**, are installed

2. Bootstrap, if required, as described in **Bootstrapping**

3. Configure the package using the `--with-dpdk` flag:

```
$ ./configure --with-dpdk=$DPDK_BUILD
```

where `DPDK_BUILD` is the path to the built DPDK library. This can be skipped if DPDK library is installed in its default location.

If no path is provided to `--with-dpdk`, but a pkg-config configuration for libdpdk is available the include paths will be generated via an equivalent `pkg-config --cflags libdpdk`.

> **Note**
>
> While `--with-dpdk` is required, you can pass any other configuration option described in **Configuring**.

4. Build and install OVS, as described in **Building**

Additional information can be found in **Open vSwitch on Linux, FreeBSD and NetBSD**.

> **Note**
>
> If you are running using the Fedora or Red Hat package, the Open vSwitch daemon will run as a non-root user. This implies that you must have a working IOMMU. Visit the **RHEL README** for additional information.

Setup

Setup Hugepages

Allocate a number of 2M Huge pages:

- For persistent allocation of huge pages, write to hugepages.conf file in */etc/sysctl.d*:

```
$ echo 'vm.nr_hugepages=2048' > /etc/sysctl.d/hugepag
```

- For run-time allocation of huge pages, use the `sysctl` utility:

```
$ sysctl -w vm.nr_hugepages=N  # where N = No. of 2M
```


To verify hugepage configuration:

```
$ grep HugePages_ /proc/meminfo
```

Mount the hugepages, if not already mounted by default:

```
$ mount -t hugetlbfs none /dev/hugepages``
```

Note

The amount of hugepage memory required can be affected by various aspects of the datapath and device configuration. Refer to **DPDK Device Memory Models** for more details.

Setup DPDK devices using VFIO

VFIO is prefered to the UIO driver when using recent versions of DPDK. VFIO support required support from both the kernel and BIOS. For the former, kernel version > 3.6 must be used. For the latter, you must enable VT-d in the BIOS and ensure this is configured via grub. To ensure VT-d is enabled via the BIOS, run:

```
$ dmesg | grep -e DMAR -e IOMMU
```

If VT-d is not enabled in the BIOS, enable it now.

To ensure VT-d is enabled in the kernel, run:

```
$ cat /proc/cmdline | grep iommu=pt
$ cat /proc/cmdline | grep intel_iommu=on
```

If VT-d is not enabled in the kernel, enable it now.

Once VT-d is correctly configured, load the required modules and bind the NIC to the VFIO driver:

```
$ modprobe vfio-pci
$ /usr/bin/chmod a+x /dev/vfio
$ /usr/bin/chmod 0666 /dev/vfio/*
$ $DPDK_DIR/usertools/dpdk-devbind.py --bind=vfio-pci eth1
$ $DPDK_DIR/usertools/dpdk-devbind.py --status
```

Setup OVS

Open vSwitch should be started as described in **Open vSwitch on Linux, FreeBSD and NetBSD** with the exception of ovs-vswitchd,

which requires some special configuration to enable DPDK functionality. DPDK configuration arguments can be passed to ovs-vswitchd via the `other_config` column of the `Open_vSwitch` table. At a minimum, the `dpdk-init` option must be set to either `true` or `try`. For example:

```
$ export PATH=$PATH:/usr/local/share/openvswitch/scripts
$ export DB_SOCK=/usr/local/var/run/openvswitch/db.sock
$ ovs-vsctl --no-wait set Open_vSwitch . other_config:dpdk
$ ovs-ctl --no-ovsdb-server --db-sock="$DB_SOCK" start
```

There are many other configuration options, the most important of which are listed below. Defaults will be provided for all values not explicitly set.

`dpdk-init`
: Specifies whether OVS should initialize and support DPDK ports. This field can either be `true` or `try`. A value of `true` will cause the ovs-vswitchd process to abort on initialization failure. A value of `try` will imply that the ovs-vswitchd process should continue running even if the EAL initialization fails.

`dpdk-lcore-mask`
: Specifies the CPU cores on which dpdk lcore threads should be spawned and expects hex string (eg '0x123').

`dpdk-socket-mem`
: Comma separated list of memory to pre-allocate from hugepages on specific sockets. If not specified, 1024 MB will be set for each numa node by default.

`dpdk-hugepage-dir`
: Directory where hugetlbfs is mounted

`vhost-sock-dir`
: Option to set the path to the vhost-user unix socket files.

If allocating more than one GB hugepage, you can configure the amount of memory used from any given NUMA nodes. For example, to use 1GB from NUMA node 0 and 0GB for all other NUMA nodes, run:

```
$ ovs-vsctl --no-wait set Open_vSwitch . \
    other_config:dpdk-socket-mem="1024,0"
```

or:

```
$ ovs-vsctl --no-wait set Open_vSwitch . \
    other_config:dpdk-socket-mem="1024"
```

Note

Changing any of these options requires restarting the ovs-vswitchd application

See the section `Performance Tuning` for important DPDK customizations.

Validating

DPDK support can be confirmed by validating the `dpdk_initialized` boolean value from the ovsdb. A value of `true` means that the DPDK EAL initialization succeeded:

```
$ ovs-vsctl get Open_vSwitch . dpdk_initialized
true
```

Additionally, the library version linked to ovs-vswitchd can be confirmed with either the ovs-vswitchd logs, or by running either of the commands:

```
$ ovs-vswitchd --version
ovs-vswitchd (Open vSwitch) 2.9.0
DPDK 17.11.0
$ ovs-vsctl get Open_vSwitch . dpdk_version
"DPDK 17.11.0"
```

At this point you can use ovs-vsctl to set up bridges and other Open vSwitch features. Seeing as we've configured the DPDK datapath, we will use DPDK-type ports. For example, to create a userspace bridge named `br0` and add two `dpdk` ports to it, run:

```
$ ovs-vsctl add-br br0 -- set bridge br0 datapath_type=net
$ ovs-vsctl add-port br0 myportnameone -- set Interface my
    type=dpdk options:dpdk-devargs=0000:06:00.0
$ ovs-vsctl add-port br0 myportnametwo -- set Interface my
    type=dpdk options:dpdk-devargs=0000:06:00.1
```

DPDK devices will not be available for use until a valid dpdk-devargs is specified.

Refer to ovs-vsctl(8) and **Using Open vSwitch with DPDK** for more details.

Performance Tuning

To achieve optimal OVS performance, the system can be configured and that includes BIOS tweaks, Grub cmdline additions, better understanding of NUMA nodes and apt selection of PCIe slots for NIC placement.

Note

This section is optional. Once installed as described above, OVS with DPDK will work out of the box.

Recommended BIOS Settings

Recommended BIOS Settings

Setting	Value
C3 Power State	Disabled
C6 Power State	Disabled
MLC Streamer	Enabled
MLC Spacial Prefetcher	Enabled
DCU Data Prefetcher	Enabled
DCA	Enabled
CPU Power and Performance	Performance
Memeory RAS and Performance Config -> NUMA optimized	Enabled

PCIe Slot Selection

The fastpath performance can be affected by factors related to the placement of the NIC, such as channel speeds between PCIe slot and CPU or the proximity of PCIe slot to the CPU cores running the DPDK application. Listed below are the steps to identify right PCIe slot.

1. Retrieve host details using `dmidecode`. For example:

```
$ dmidecode -t baseboard | grep "Product Name"
```

2. Download the technical specification for product listed, e.g: S2600WT2

3. Check the Product Architecture Overview on the Riser slot placement, CPU sharing info and also PCIe channel speeds

   For example: On S2600WT, CPU1 and CPU2 share Riser Slot 1 with Channel speed between CPU1 and Riser Slot1 at 32GB/s, CPU2 and Riser Slot1 at 16GB/s. Running DPDK app on CPU1 cores and NIC inserted in to Riser card Slots will optimize OVS performance in this case.

4. Check the Riser Card #1 - Root Port mapping information, on the available slots and individual bus speeds. In S2600WT slot 1, slot 2 has high bus speeds and are potential slots for NIC placement.

Advanced Hugepage Setup

Allocate and mount 1 GB hugepages.

- For persistent allocation of huge pages, add the following options to the kernel bootline:

```
default_hugepagesz=1GB hugepagesz=1G hugepages=N
```

For platforms supporting multiple huge page sizes, add multiple options:

```
default_hugepagesz=<size> hugepagesz=<size> hugepages
```

where:

`N`

number of huge pages requested

`size`

huge page size with an optional suffix `[kKmMgG]`

- For run-time allocation of huge pages:

```
$ echo N > /sys/devices/system/node/nodeX/hugepages/h
```

where:

`N`

number of huge pages requested

`X`

NUMA Node

Note

For run-time allocation of 1G huge pages, Contiguous Memory Allocator (`CONFIG_CMA`) has to be supported by kernel, check your Linux distro.

Now mount the huge pages, if not already done so:

```
$ mount -t hugetlbfs -o pagesize=1G none /dev/hugepages
```

Isolate Cores

The `isolcpus` option can be used to isolate cores from the Linux scheduler. The isolated cores can then be used to dedicatedly run HPC applications or threads. This helps in better application performance due to zero context switching and minimal cache thrashing. To run platform logic on core 0 and isolate cores between 1 and 19 from scheduler, add `isolcpus=1-19` to GRUB cmdline.

Note

It has been verified that core isolation has minimal advantage due to mature Linux scheduler in some circumstances.

Compiler Optimizations

The default compiler optimization level is `-O2`. Changing this to more aggressive compiler optimization such as `-O3 -march=native` with gcc (verified on 5.3.1) can produce performance gains though not significant. `-march=native` will produce optimized code on local machine and should be used when software compilation is done on Testbed.

Multiple Poll-Mode Driver Threads

With pmd multi-threading support, OVS creates one pmd thread for each NUMA node by default, if there is at least one DPDK interface from that NUMA node added to OVS. However, in cases where there are multiple ports/rxq's producing traffic, performance can be improved by creating multiple pmd threads running on separate cores. These pmd threads can share the workload by each being responsible for different ports/rxq's. Assignment of ports/rxq's to pmd threads is done automatically.

A set bit in the mask means a pmd thread is created and pinned to the corresponding CPU core. For example, to run pmd threads on core 1 and 2:

```
$ ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0
```


When using dpdk and dpdkvhostuser ports in a bi-directional VM loopback as shown below, spreading the workload over 2 or 4 pmd

threads shows significant improvements as there will be more total CPU occupancy available:

```
NIC port0 <-> OVS <-> VM <-> OVS <-> NIC port 1
```

Refer to ovs-vswitchd.conf.db(5) for additional information on configuration options.

Affinity

For superior performance, DPDK pmd threads and Qemu vCPU threads needs to be affinitized accordingly.

- PMD thread Affinity

  A poll mode driver (pmd) thread handles the I/O of all DPDK interfaces assigned to it. A pmd thread shall poll the ports for incoming packets, switch the packets and send to tx port. A pmd thread is CPU bound, and needs to be affinitized to isolated cores for optimum performance. Even though a PMD thread may exist, the thread only starts consuming CPU cycles if there is at least one receive queue assigned to the pmd.

  > **Note**
  >
  > On NUMA systems, PCI devices are also local to a NUMA node. Unbound rx queues for a PCI device will be assigned to a pmd on it's local NUMA node if a non-isolated PMD exists on that NUMA node. If not, the queue will be assigned to a non-isolated pmd on a remote NUMA node. This will result in reduced maximum throughput on that device and possibly on other devices assigned to that pmd thread. If such a queue assignment is made a warning message will be logged: "There's no available (non-isolated) pmd thread on numa node N. Queue Q on port P will be assigned to the pmd on core C (numa node N'). Expect reduced performance."

  Binding PMD threads to cores is described in the above section `Multiple Poll-Mode Driver Threads`.

- QEMU vCPU thread Affinity

  A VM performing simple packet forwarding or running complex packet pipelines has to ensure that the vCPU threads performing the work has as much CPU occupancy as possible.

  For example, on a multicore VM, multiple QEMU vCPU threads shall be spawned. When the DPDK `testpmd` application that does packet forwarding is invoked, the `taskset` command should be used to affinitize the vCPU threads to the dedicated isolated cores on the host system.

Enable HyperThreading

With HyperThreading, or SMT, enabled, a physical core appears as two logical cores. SMT can be utilized to spawn worker threads on logical cores of the same physical core there by saving additional cores.

With DPDK, when pinning pmd threads to logical cores, care must be taken to set the correct bits of the `pmd-cpu-mask` to ensure that the pmd threads are pinned to SMT siblings.

Take a sample system configuration, with 2 sockets, 2 * 10 core processors, HT enabled. This gives us a total of 40 logical cores. To identify the physical core shared by two logical cores, run:

```
$ cat /sys/devices/system/cpu/cpuN/topology/thread_sibling
```

where `N` is the logical core number.

In this example, it would show that cores `1` and `21` share the same physical core. Logical cores can be specified in pmd-cpu-masks similarly to physical cores, as described in `Multiple Poll-Mode Driver Threads`.

NUMA/Cluster-on-Die

Ideally inter-NUMA datapaths should be avoided where possible as packets will go across QPI and there may be a slight performance penalty when compared with intra NUMA datapaths. On Intel Xeon Processor E5 v3, Cluster On Die is introduced on models that have 10 cores or more. This makes it possible to logically split a socket into two NUMA regions and again it is preferred where possible to keep critical datapaths within the one cluster.

It is good practice to ensure that threads that are in the datapath are pinned to cores in the same NUMA area. e.g. pmd threads and QEMU vCPUs responsible for forwarding. If DPDK is built with `CONFIG_RTE_LIBRTE_VHOST_NUMA=y`, vHost User ports automatically detect the NUMA socket of the QEMU vCPUs and will be serviced by a PMD from the same node provided a core on this node is enabled in the `pmd-cpu-mask`. `libnuma` packages are required for this feature.

Binding PMD threads is described in the above section `Multiple Poll-Mode Driver Threads`.

DPDK Physical Port Rx Queues


```
$ ovs-vsctl set Interface <DPDK interface> options:n_rxq=<
```

The above command sets the number of rx queues for DPDK physical interface. The rx queues are assigned to pmd threads on the same NUMA node in a round-robin fashion.

DPDK Physical Port Queue Sizes

```
$ ovs-vsctl set Interface dpdk0 options:n_rxq_desc=<intege
$ ovs-vsctl set Interface dpdk0 options:n_txq_desc=<intege
```

The above command sets the number of rx/tx descriptors that the NIC associated with dpdk0 will be initialised with.

Different `n_rxq_desc` and `n_txq_desc` configurations yield different benefits in terms of throughput and latency for different scenarios. Generally, smaller queue sizes can have a positive impact for latency at the expense of throughput. The opposite is often true for larger queue sizes. Note: increasing the number of rx descriptors eg. to 4096 may have a negative impact on performance due to the fact that non-vectorised DPDK rx functions may be used. This is dependent on the driver in use, but is true for the commonly used i40e and ixgbe DPDK drivers.

Exact Match Cache

Each pmd thread contains one Exact Match Cache (EMC). After initial flow setup in the datapath, the EMC contains a single table and provides the lowest level (fastest) switching for DPDK ports. If there is a miss in the EMC then the next level where switching will occur is the datapath classifier. Missing in the EMC and looking up in the datapath classifier incurs a significant performance penalty. If lookup misses occur in the EMC because it is too small to handle the number of flows, its size can be increased. The EMC size can be modified by editing the define `EM_FLOW_HASH_SHIFT` in `lib/dpif-netdev.c`.

As mentioned above, an EMC is per pmd thread. An alternative way of increasing the aggregate amount of possible flow entries in EMC and avoiding datapath classifier lookups is to have multiple pmd threads running.

Rx Mergeable Buffers

Rx mergeable buffers is a virtio feature that allows chaining of multiple virtio descriptors to handle large packet sizes. Large packets are handled by reserving and chaining multiple free descriptors together. Mergeable buffer support is negotiated between the virtio driver and virtio device and is supported by the DPDK vhost library. This behavior is supported and enabled by default, however in the case where the user knows that rx mergeable buffers are not needed i.e. jumbo frames are not needed, it can be forced off by adding `mrg_rxbuf=off` to the

QEMU command line options. By not reserving multiple chains of descriptors it will make more individual virtio descriptors available for rx to the guest using dpdkvhost ports and this can improve performance.

Output Packet Batching

To make advantage of batched transmit functions, OVS collects packets in intermediate queues before sending when processing a batch of received packets. Even if packets are matched by different flows, OVS uses a single send operation for all packets destined to the same output port.

Furthermore, OVS is able to buffer packets in these intermediate queues for a configurable amount of time to reduce the frequency of send bursts at medium load levels when the packet receive rate is high, but the receive batch size still very small. This is particularly beneficial for packets transmitted to VMs using an interrupt-driven virtio driver, where the interrupt overhead is significant for the OVS PMD, the host operating system and the guest driver.

The `tx-flush-interval` parameter can be used to specify the time in microseconds OVS should wait between two send bursts to a given port (default is `0`). When the intermediate queue fills up before that time is over, the buffered packet batch is sent immediately:

```
$ ovs-vsctl set Open_vSwitch . other_config:tx-flush-inter
```

This parameter influences both throughput and latency, depending on the traffic load on the port. In general lower values decrease latency while higher values may be useful to achieve higher throughput.

Low traffic (`packet rate < 1 / tx-flush-interval`) should not experience any significant latency or throughput increase as packets are forwarded immediately.

At intermediate load levels (`1 / tx-flush-interval < packet rate < 32 / tx-flush-interval`) traffic should experience an average latency increase of up to `1 / 2 * tx-flush-interval` and a possible throughput improvement.

Very high traffic (`packet rate >> 32 / tx-flush-interval`) should experience the average latency increase equal to `32 / (2 * packet rate)`. Most send batches in this case will contain the maximum number of packets (`32`).

A `tx-burst-interval` value of `50` microseconds has shown to provide a good performance increase in a `PHY-VM-PHY` scenario on `x86` system for interrupt-driven guests while keeping the latency increase at a reasonable level:

https://mail.openvswitch.org/pipermail/ovs-dev/2017-December/341628.html

Note

Throughput impact of this option significantly depends on the scenario and the traffic patterns. For example: `tx-burst-interval` value of `50` microseconds shows performance degradation in `PHY-VM-PHY` with bonded PHY scenario while testing with `256 - 1024` packet flows:

https://mail.openvswitch.org/pipermail/ovs-dev/2017-December/341700.html

The average number of packets per output batch can be checked in PMD stats:

```
$ ovs-appctl dpif-netdev/pmd-stats-show
```

Limitations

- Network Interface Firmware requirements: Each release of DPDK is validated against a specific firmware version for a supported Network Interface. New firmware versions introduce bug fixes, performance improvements and new functionality that DPDK leverages. The validated firmware versions are available as part of the release notes for DPDK. It is recommended that users update Network Interface firmware to match what has been validated for the DPDK release.

  The latest list of validated firmware versions can be found in the **DPDK release notes**.

- Upper bound MTU: DPDK device drivers differ in how the L2 frame for a given MTU value is calculated e.g. i40e driver includes 2 x vlan headers in MTU overhead, em driver includes 1 x vlan header, ixgbe driver does not include a vlan header in overhead. Currently it is not possible for OVS DPDK to know what upper bound MTU value is supported for a given device. As such OVS DPDK must provision for the case where the L2 frame for a given MTU includes 2 x vlan headers. This reduces the upper bound MTU value for devices that do not include vlan headers in their L2 frames by 8 bytes e.g. ixgbe devices upper bound MTU is reduced from 9710 to 9702. This work around is temporary and is expected to be removed once a method is provided by DPDK to query the upper bound MTU value for a given device.

Reporting Bugs

Report problems to **bugs@openvswitch.org**.

Distributions packaging Open vSwitch ›
‹ Open vSwitch without Kernel Support