Compression Device Drivers

Release 18.08.1

April 02, 2019

CHAPTER

ONE

COMPRESSION DEVICE SUPPORTED FUNCTIONALITY MATRICES

1.1 Supported Feature Flags

Table 1.1: Features availability in compression drivers

Feature	isal	octeontx	qat	zlib
HW Accelerated		Y	Y	
CPU SSE	Y			
CPU AVX	Y			
CPU AVX2	Y			
CPU AVX512	Y			
CPU NEON				
Stateful				
Pass-through				Y
OOP SGL In SGL Out	Y		Y	
OOP SGL In LB Out	Y		Y	
OOP LB In SGL Out	Y		Y	
Deflate	Y	Y	Y	Y
LZS				
Adler32			Y	
Crc32			Y	
Adler32&Crc32			Y	
Fixed	Y	Y	Y	Y
Dynamic	Y	Y		Y

Note:

- "Pass-through" feature flag refers to the ability of the PMD to let input buffers pass-through it, copying the input to the output, without making any modifications to it (no compression done).
- "OOP SGL In SGL Out" feature flag stands for "Out-of-place Scatter-gather list Input, Scatter-gater list Output", which means PMD supports different scatter-gather styled input and output buffers (i.e. both can consists of multiple segments).
- "OOP SGL In LB Out" feature flag stands for "Out-of-place Scatter-gather list Input, Linear Buffers Output", which means PMD supports input from scatter-gathered styled buffers, outputting linear buffers (i.e. single segment).
- "OOP LB In SGL Out" feature flag stands for "Out-of-place Linear Buffers Input, Scatter-gather list Output", which means PMD supports input from linear buffer, outputting

scatter-gathered styled buffers.

CHAPTER

TWO

ISA-L COMPRESSION POLL MODE DRIVER

The ISA-L PMD (**librte_pmd_isal_comp**) provides poll mode compression & decompression driver support for utilizing Intel ISA-L library, which implements the deflate algorithm for both Deflate(compression) and Inflate(decompression).

2.1 Features

ISA-L PMD has support for:

Compression/Decompression algorithm:

- DEFLATE

Huffman code type:

- FIXED
- DYNAMIC

Window size support:

- 32K

Level guide:

The ISA-L levels have been mapped to somewhat correspond to the same ZLIB level, i.e. ZLIB L1 gives a compression ratio similar to ISA-L L1. Compressdev level 0 enables "No Compression", which passes the uncompressed data to the output buffer, plus deflate headers. The ISA-L library does not support this, therefore compressdev level 0 is not supported.

The compressdev API has 10 levels, 0-9. ISA-L has 4 levels of compression, 0-3. As a result the level mappings from the API to the PMD are shown below.

Table 2.1: Level mapping from Compressdev to ISA-L PMD.

Compressdev API Level	PMD Functionality	Internal ISA-L Level
0	No compression, Not Supported	—
1	Dynamic (Fast compression)	1
2	Dynamic (Higher compression ratio)	2
3	Dynamic (Best compression ratio)	3 (Level 2 if no AVX512/AVX2)
4	Dynamic (Best compression ratio)	Same as above
5	Dynamic (Best compression ratio)	Same as above
6	Dynamic (Best compression ratio)	Same as above
7	Dynamic (Best compression ratio)	Same as above
8	Dynamic (Best compression ratio)	Same as above
9	Dynamic (Best compression ratio)	Same as above

Note: The above table only shows mapping when API calls for dynamic compression. For fixed compression, regardless of API level, internally ISA-L level 0 is always used.

2.2 Limitations

- Compressdev level 0, no compression, is not supported.
- Checksums will not be supported until future release.

2.3 Installation

- To build DPDK with Intel's ISA-L library, the user is required to download the library from https://github.com/01org/isa-l.
- Once downloaded, the user needs to build the library, the ISA-L autotools are usually sufficient:

```
./autogen.sh
./configure
```

- make can be used to install the library on their system, before building DPDK:

```
make
sudo make install
```

- To build with meson, the **libisal.pc** file, must be copied into "pkgconfig", e.g. /usr/lib/pkgconfig or /usr/lib64/pkgconfig depending on your system, for meson to find the ISA-L library. The **libisal.pc** is located in library sources:

```
cp isal/libisal.pc /usr/lib/pkgconfig/
```

2.4 Initialization

In order to enable this virtual compression PMD, user must:

- Set `CONFIG_RTE_LIBRTE_PMD_ISAL=y` in config/common_base.

To use the PMD in an application, user must:

- Call `rte_vdev_init("compress_isal")` within the application.
- Use `--vdev="compress_isal"` in the EAL options, which will call `rte_vdev_init()` internally.

The following parameter (optional) can be provided in the previous two calls:

- `socket_id:` Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).

CHAPTER

THREE

OCTEONTX ZIP COMPRESSION POLL MODE DRIVER

The Octeontx ZIP PMD (**librte_pmd_octeontx_zip**) provides poll mode compression & decompression driver for ZIP HW offload device, found in **Cavium OCTEONTX** SoC family.

More information can be found at Cavium, Inc Official Website.

3.1 Features

Octeontx ZIP PMD has support for:

Compression/Decompression algorithm:

- DEFLATE

Huffman code type:

- FIXED
- DYNAMIC

Window size support:

- 2 to 2^14

3.2 Limitations

- Chained mbufs are not supported.

3.3 Supported OCTEONTX SoCs

- CN83xx

3.4 Steps To Setup Platform

> Octeontx SDK includes kernel image which provides Octeontx ZIP PF driver to manage configuration of ZIPVF device Required version of SDK is "OCTEONTX-SDK-6.2.0-build35" or above.

SDK can be install by using below command. #rpm -ivh CTEONTX-SDK-6.2.0-build35.x86_64.rpm –force –nodeps It will install OCTEONTX-SDK at following default location /usr/local/Cavium_Networks/OCTEONTX-SDK/

For more information on building and booting linux kernel on OCTEONTX please refer /usr/local/Cavium_Networks/OCTEONTX-SDK/docs/OcteonTX-SDK-UG_6.2.0.pdf.

SDK and related information can be obtained from: Cavium support site.

3.5 Installation

3.5.1 Driver Compilation

To compile the OCTEONTX ZIP PMD for Linux arm64 gcc target, run the following `make` command:

```
cd <DPDK-source-directory>
make config T=arm64-thunderx-linuxapp-gcc install
```

3.6 Initialization

The octeontx zip is exposed as pci device which consists of a set of PCIe VF devices. On EAL initialization, ZIP PCIe VF devices will be probed. To use the PMD in an application, user must:

- run dev_bind script to bind eight ZIP PCIe VFs to the `vfio-pci` driver:

  ```
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:00.1
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:00.2
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:00.3
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:00.4
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:00.5
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:00.6
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:00.7
  ./usertools/dpdk-devbind.py -b vfio-pci 0001:04:01.0
  ```

- The unit test cases can be tested as below:

  ```
  reserve enough huge pages
  cd to the top-level DPDK directory
  export RTE_TARGET=arm64-thunderx-linuxapp-gcc
  export RTE_SDK=`pwd`
  cd to test/test
  type the command "make" to compile
  run the tests with "./test"
  type the command "compressdev_autotest" to test
  ```

CHAPTER

FOUR

INTEL(R) QUICKASSIST (QAT) COMPRESSION POLL MODE DRIVER

The QAT compression PMD provides poll mode compression & decompression driver support for the following hardware accelerator devices:

- `Intel QuickAssist Technology C62x`
- `Intel QuickAssist Technology C3xxx`

4.1 Features

QAT compression PMD has support for:

Compression/Decompression algorithm:

- DEFLATE

Huffman code type:

- FIXED

Window size support:

- 32K

Checksum generation:

- CRC32, Adler and combined checksum

4.2 Limitations

- Compressdev level 0, no compression, is not supported.
- Dynamic Huffman encoding is not yet supported.

4.3 Installation

The QAT compression PMD is built by default with a standard DPDK build.

It depends on a QAT kernel driver, see qat_kernel_installation.

CHAPTER

FIVE

ZLIB COMPRESSION POLL MODE DRIVER

The ZLIB PMD (**librte_pmd_zlib**) provides poll mode compression & decompression driver based on SW zlib library,

5.1 Features

ZLIB PMD has support for:

Compression/Decompression algorithm:

- DEFLATE

Huffman code type:

- FIXED
- DYNAMIC

Window size support:

- Min - 256 bytes
- Max - 32K

5.2 Limitations

- Scatter-Gather and Stateful not supported.

5.3 Installation

- To build DPDK with ZLIB library, the user is required to download the `libz` library.
- Use following command for installation.
- **For Fedora users::** sudo yum install zlib-devel
- **For Ubuntu users::** sudo apt-get install zlib1g-dev
- Once downloaded, the user needs to build the library.
- To build from sources download zlib sources from http://zlib.net/ and do following before building DPDK:

```
make
sudo make install
```

5.4 Initialization

In order to enable this virtual compression PMD, user must:

- Set `CONFIG_RTE_LIBRTE_PMD_ZLIB=y` in config/common_base.

To use the PMD in an application, user must:

- Call `rte_vdev_init("compress_zlib")` within the application.
- Use `--vdev="compress_zlib"` in the EAL options, which will call `rte_vdev_init()` internally.

The following parameter (optional) can be provided in the previous two calls:

- `socket_id:` Specify the socket where the memory for the device is going to be allocated (by default, socket_id will be the socket where the core that is creating the PMD is running on).