DPDKCap

DPDKCap is packet capture tool based on DPDK. It provides a multi-port, multi-core optimized capture with on the fly compression. Thus particularly suiting captures at very high speeds (more than 10Gpbs).

Build status

Branch	Status
Master	build passing (https://travis-ci.org/dpdkcap/dpdkcap)
Develop	build passing (https://travis-ci.org/dpdkcap/dpdkcap)

1. Installation and platform configuration

1.1 Install DPDK

Please DPDK installation instruction, either from the DPDK quick start instructions (http://dpdk.org/doc/quick-start) or from your operating system specific Getting started guide (http://dpdk.org/doc/guides/linux_gsg/build_dpdk.html).

1.2 Install dependencies

DPDKCap requires the following dependencies to be built:

- libncurses-dev

1.3 Build and Install DPDKCap

To build DPDKCap, you first need to set RTE_SDK and RTE_TARGET.

```
$ export RTE_SDK=... # Replace by your DPDK install directory
$ export RTE_TARGET=x86_64-native-linuxapp-gcc # Replace by your target
```

To build DPDKCap, run the following command into DPDKCap root directory:

```
$ make
```

2. Usage

DPDKCap works as a standard DPDK application. Thus it needs Environment Abstraction Layer (EAL) arguments before dpdkcap specific ones:

```
# ./build/dpdkcap [EAL args] -- [dpdkcap args]
```

Check out the dpdk documentation (http://dpdk.org/doc/guides/index.html) for more information on EAL arguments. You will probably need the `-l` option for cores allocation and the `--huge-dir` one for providing huge pages directory.

To get a list of DPDKCap specific available options, run:

```
# ./build/dpdkcap [EAL args] -- --help
```

2.1 Selecting cores for capture

From the available ports detected by DPDK, you can select ports to capture by using the `-p, --portmask` option. This option takes as argument an hexadecimal mask whose bits represent each port. By default, DPDKCap uses only the first port (portmask=0x1).

For example, if you want to capture ports 1, 2 and 4, use: `--portmask 0xb`

2.2 Assigning tasks to lcores

DPDKCap assigns two different tasks to lcores:

- Capturing cores enqueue packets from Ethernet ports queues into a main buffer. Each captured port must be assigned at least a core.
- Writing cores extract packets from this buffer and write them into LZO compressed pcap capture files. Each writing core writes into a different file.

As a consequence, DPDKCap needs, at least, a single writing core and as many capturing cores as ports you want to capture. Finally, a last lcore must be kept to handle logs and statistics. However, depending on your traffic bandwidth and your system capabilities, you might need to use more cores.

The `-c, --per_port_c_cores` option allocates `NB_CORES_PER_PORT` capturing cores **per selected port**.

The `-w, --num_w_cores` option allocates a **total** of `NB_CORES` writing cores.

Note that the writing task requires more computational power than the capture one (due to compression), thus you will probably need to allow more writing cores than capture ones. This being said, size your storage system accordingly, as thousands cores could not achieve a full capture with a too low storage system bandwidth.

2.3 Limiting file size or duration

Depending on the data you want to capture, you might need to split the capture into several files. Two options are available to limit file size/duration:

- The `-G, --rotate_seconds` option creates a new file every T seconds.
- The `-C, --limit_file_size` option creates a new file when the current file size goes over the specified SIZE.

You can specify the output file template using the `-o, --output` option. This is necessary with the `-G, --rotate_seconds` option if you do not want to erase the same file again and again. See the following section.

2.4 Setting output template

The `-o,--output` let you provide a template for the output file. This template is formatted according to the following tokens:

- `%COREID` this is replaced by the writing core id into the filename. This token is mandatory and will be automatically appended to the output file template if not present.
- `%FCOUNT` this is replaced by a counter that allows distinguishing files created by the `-C, --limit_file_size` option. If this option is used, this token is mandatory and will be automatically appended to the output file template if not present.
- Date *strftime* tokens. These tokens are replaced according to *strftime* standard. This date is updated every time the `-G, --rotate_seconds` option triggers a file change. These tokens are not mandatory with this option, but you might overwrite previously created files.

2.5 Other options

- `-s, --snaplen` limits the packet capture to LENGTH bytes.
- `-S, --statistics` prints a set of statistics while the capture is running.
- `--logs` output logs into the specified file instead of stderr.
- `--no-compression` disables the LZO compression. This is not advised, as it greatly increase the disk I/O. It can however be used for capturing low speed traffic.
- `-m, --num_mbufs` changes the number of memory buffers used by dpdkcap. Note that the default value might not work in your situation (mbufs pool allocation failure at startup or RX mbufs allocation failures while running). Optimal values (in term of memory usage) are powers of 2 minus one (n=2^q-1).
- `-d, --rx_desc` allow you to fix the number of RX descriptors per queue used. This value can be fixed in a per port fashion. The following formats are available:
    - A single integer value: fixes the given number of RX descriptors for all ports.
    - A list of key-values, assigning a value to the given port id, following this format:

```
<matrix>   := <key>.<nb_rx_desc> { "," <key>.<nb_rx_desc> "," ...}
<key>      := { <interval> | <port> }
<interval> := <lower_port> "-" <upper_port>
```

      Examples:

```
512               - all ports have 512 RX desc per queue
0.256, 1.512      - port 0 has 256 RX desc per queue,
                    port 1 has 512 RX desc per queue
0-2.256, 3.1024   - ports 0, 1 and 2 have 256 RX desc per queue,
                    port 3 has 1024 RX desc per queue
```

- `--pcapng` replaces the *libpcap* output format by the *pcapng* (next generation) one. See its specification here (https://github.com/pcapng/pcapng).

3. Troubleshooting

Here is a list of common issues and how to solve them:

- Mbufs pool allocation failure: try to reduce the number of memory buffers used with the `-m, -num_mbufs` option.
- Mbufs allocation failures (while running): try to raise the number of memory buffers used with the `-m, -num_mbufs` option.
- Problems with with RX queues configuration: the default number of RX descriptors configured might be too high for your interface. Try to change the number of RX descriptors used with the `-d, --rx_desc` option.

4. Software License Agreements

DPDKCap is distributed under the BSD License, see LICENSE.txt.