DPDK

DATA PLANE DEVELOPMENT KIT

Testpmd Application User Guide

Release 18.08.1

April 02, 2019

CONTENTS

CHAPTER

ONE

INTRODUCTION

This document is a user guide for the `testpmd` example application that is shipped as part of the Data Plane Development Kit.

The `testpmd` application can be used to test the DPDK in a packet forwarding mode and also to access NIC hardware features such as Flow Director. It also serves as a example of how to build a more fully-featured application using the DPDK SDK.

The guide shows how to build and run the testpmd application and how to configure the application from the command line and the run-time environment.

CHAPTER

TWO

COMPILING THE APPLICATION

The `testpmd` application is compiled as part of the main compilation of the DPDK libraries and tools. Refer to the DPDK Getting Started Guides for details. The basic compilation steps are:

1. Set the required environmental variables and go to the source directory:

   ```
   export RTE_SDK=/path/to/rte_sdk
   cd $RTE_SDK
   ```

2. Set the compilation target. For example:

   ```
   export RTE_TARGET=x86_64-native-linuxapp-gcc
   ```

3. Build the application:

   ```
   make install T=$RTE_TARGET
   ```

   The compiled application will be located at:

   ```
   $RTE_SDK/$RTE_TARGET/app/testpmd
   ```

CHAPTER

THREE

RUNNING THE APPLICATION

3.1 EAL Command-line Options

The following are the EAL command-line options that can be used in conjunction with the testpmd, or any other DPDK application. See the DPDK Getting Started Guides for more information on these options.

- `-c COREMASK`

  Set the hexadecimal bitmask of the cores to run on.

- `-l CORELIST`

  List of cores to run on

  The argument format is `<c1>[-c2][,c3[-c4],...]` where `c1`, `c2`, etc are core indexes between 0 and 128.

- `--lcores COREMAP`

  Map lcore set to physical cpu set

  The argument format is:

  ```
  <lcores[@cpus]>[<,lcores[@cpus]>...]
  ```

  Lcore and CPU lists are grouped by `(` and `)` Within the group. The `-` character is used as a range separator and `,` is used as a single number separator. The grouping `()` can be omitted for single element group. The `@` can be omitted if cpus and lcores have the same value.

Note: At a given instance only one core option `--lcores`, `-l` or `-c` can be used.

- `--master-lcore ID`

  Core ID that is used as master.

- `-n NUM`

  Set the number of memory channels to use.

- `-b, --pci-blacklist domain:bus:devid.func`

  Blacklist a PCI device to prevent EAL from using it. Multiple -b options are allowed.

- `-d LIB.so`

  Load an external driver. Multiple -d options are allowed.

- `-w, --pci-whitelist domain:bus:devid:func`

  Add a PCI device in white list.

- `-m MB`

  Memory to allocate. See also `--socket-mem`.

- `-r NUM`

  Set the number of memory ranks (auto-detected by default).

- `-v`

  Display the version information on startup.

- `--syslog`

  Set the syslog facility.

- `--socket-mem`

  Set the memory to allocate on specific sockets (use comma separated values).

- `--huge-dir`

  Specify the directory where the hugetlbfs is mounted.

- `mbuf-pool-ops-name`:

  Pool ops name for mbuf to use.

- `--proc-type`

  Set the type of the current process.

- `--file-prefix`

  Prefix for hugepage filenames.

- `-vmware-tsc-map`

  Use VMware TSC map instead of native RDTSC.

- `--vdev`

  Add a virtual device using the format:

  ```
  <driver><id>[,key=val, ...]
  ```

  For example:

  ```
  --vdev 'net_pcap0,rx_pcap=input.pcap,tx_pcap=output.pcap'
  ```

- `--base-virtaddr`

  Specify base virtual address.

- `--create-uio-dev`

  Create `/dev/uioX` (usually done by hotplug).

- `--no-shconf`

  No shared config (mmap-ed files).

- `--no-pci`

  Disable pci.

- `--no-hpet`

  Disable hpet.

- `--no-huge`

  Use malloc instead of hugetlbfs.

3.2 Testpmd Command-line Options

The following are the command-line options for the testpmd applications. They must be separated from the EAL options, shown in the previous section, with a `--` separator:

```
sudo ./testpmd -l 0-3 -n 4 -- -i --portmask=0x1 --nb-cores=2
```

The commandline options are:

- `-i, --interactive`

  Run testpmd in interactive mode. In this mode, the testpmd starts with a prompt that can be used to start and stop forwarding, configure the application and display stats on the current packet processing session. See *Testpmd Runtime Functions* for more details.

  In non-interactive mode, the application starts with the configuration specified on the command-line and immediately enters forwarding mode.

- `-h, --help`

  Display a help message and quit.

- `-a, --auto-start`

  Start forwarding on initialization.

- `--tx-first`

  Start forwarding, after sending a burst of packets first.

Note: This flag should be only used in non-interactive mode.

- `--stats-period PERIOD`

  Display statistics every PERIOD seconds, if interactive mode is disabled. The default value is 0, which means that the statistics will not be displayed.

- `--nb-cores=N`

  Set the number of forwarding cores, where 1 <= N <= “number of cores” or `CONFIG_RTE_MAX_LCORE` from the configuration file. The default value is 1.

- `--nb-ports=N`

  Set the number of forwarding ports, where 1 <= N <= “number of ports” on the board or `CONFIG_RTE_MAX_ETHPORTS` from the configuration file. The default value is the number of ports on the board.

- `--coremask=0xXX`

  Set the hexadecimal bitmask of the cores running the packet forwarding test. The master lcore is reserved for command line parsing only and cannot be masked on for packet forwarding.

- `--portmask=0xXX`

  Set the hexadecimal bitmask of the ports used by the packet forwarding test.

- `--numa`

  Enable NUMA-aware allocation of RX/TX rings and of RX memory buffers (mbufs). [Default setting]

- `--no-numa`

  Disable NUMA-aware allocation of RX/TX rings and of RX memory buffers (mbufs).

- `--port-numa-config=(port,socket)[,(port,socket)]`

  Specify the socket on which the memory pool to be used by the port will be allocated.

- `--ring-numa-config=(port,flag,socket)[,(port,flag,socket)]`

  Specify the socket on which the TX/RX rings for the port will be allocated. Where flag is 1 for RX, 2 for TX, and 3 for RX and TX.

- `--socket-num=N`

  Set the socket from which all memory is allocated in NUMA mode, where 0 <= N < number of sockets on the board.

- `--mbuf-size=N`

  Set the data size of the mbufs used to N bytes, where N < 65536. The default value is 2048.

- `--total-num-mbufs=N`

  Set the number of mbufs to be allocated in the mbuf pools, where N > 1024.

- `--max-pkt-len=N`

  Set the maximum packet size to N bytes, where N >= 64. The default value is 1518.

- `--eth-peers-configfile=name`

  Use a configuration file containing the Ethernet addresses of the peer ports. The configuration file should contain the Ethernet addresses on separate lines:

  ```
  XX:XX:XX:XX:XX:01
  XX:XX:XX:XX:XX:02
  ...
  ```

- `--eth-peer=N,XX:XX:XX:XX:XX:XX`

  Set the MAC address `XX:XX:XX:XX:XX:XX` of the peer port N, where 0 <= N < `CONFIG_RTE_MAX_ETHPORTS` from the configuration file.

- `--pkt-filter-mode=mode`

  Set Flow Director mode where mode is either `none` (the default), `signature` or `perfect`. See *flow_director_filter* for more details.

- `--pkt-filter-report-hash=mode`

  Set Flow Director hash match reporting mode where mode is `none`, `match` (the default) or `always`.

- `--pkt-filter-size=N`

  Set Flow Director allocated memory size, where N is 64K, 128K or 256K. Sizes are in kilobytes. The default is 64.

- `--pkt-filter-flexbytes-offset=N`

  Set the flexbytes offset. The offset is defined in words (not bytes) counted from the first byte of the destination Ethernet MAC address, where N is 0 <= N <= 32. The default value is 0x6.

- `--pkt-filter-drop-queue=N`

  Set the drop-queue. In perfect filter mode, when a rule is added with queue = -1, the packet will be enqueued into the RX drop-queue. If the drop-queue does not exist, the packet is dropped. The default value is N=127.

- `--disable-crc-strip`

  Disable hardware CRC stripping.

- `--enable-lro`

  Enable large receive offload.

- `--enable-rx-cksum`

  Enable hardware RX checksum offload.

- `--enable-scatter`

  Enable scatter (multi-segment) RX.

- `--enable-hw-vlan`

  Enable hardware VLAN.

- `--enable-hw-vlan-filter`

  Enable hardware VLAN filter.

- `--enable-hw-vlan-strip`

  Enable hardware VLAN strip.

- `--enable-hw-vlan-extend`

  Enable hardware VLAN extend.

- `--enable-drop-en`

  Enable per-queue packet drop for packets with no descriptors.

- `--disable-rss`

  Disable RSS (Receive Side Scaling).

- `--port-topology=mode`

  Set port topology, where mode is `paired` (the default) or `chained`.

  In `paired` mode, the forwarding is between pairs of ports, for example: (0,1), (2,3), (4,5).

  In `chained` mode, the forwarding is to the next available port in the port mask, for example: (0,1), (1,2), (2,0).

The ordering of the ports can be changed using the portlist testpmd runtime function.

- `--forward-mode=mode`

  Set the forwarding mode where `mode` is one of the following:

  ```
  io (the default)
  mac
  macswap
  flowgen
  rxonly
  txonly
  csum
  icmpecho
  ieee1588
  tm
  ```

- `--rss-ip`

  Set RSS functions for IPv4/IPv6 only.

- `--rss-udp`

  Set RSS functions for IPv4/IPv6 and UDP.

- `--rxq=N`

  Set the number of RX queues per port to N, where 1 <= N <= 65535. The default value is 1.

- `--rxd=N`

  Set the number of descriptors in the RX rings to N, where N > 0. The default value is 128.

- `--txq=N`

  Set the number of TX queues per port to N, where 1 <= N <= 65535. The default value is 1.

- `--txd=N`

  Set the number of descriptors in the TX rings to N, where N > 0. The default value is 512.

- `--burst=N`

  Set the number of packets per burst to N, where 1 <= N <= 512. The default value is 32. If set to 0, driver default is used if defined. Else, if driver default is not defined, default of 32 is used.

- `--mbcache=N`

  Set the cache of mbuf memory pools to N, where 0 <= N <= 512. The default value is 16.

- `--rxpt=N`

  Set the prefetch threshold register of RX rings to N, where N >= 0. The default value is 8.

- `--rxht=N`

  Set the host threshold register of RX rings to N, where N >= 0. The default value is 8.

- `--rxfreet=N`

Set the free threshold of RX descriptors to N, where 0 <= N < value of –rxd. The default value is 0.

- `--rxwt=N`

  Set the write-back threshold register of RX rings to N, where N >= 0. The default value is 4.

- `--txpt=N`

  Set the prefetch threshold register of TX rings to N, where N >= 0. The default value is 36.

- `--txht=N`

  Set the host threshold register of TX rings to N, where N >= 0. The default value is 0.

- `--txwt=N`

  Set the write-back threshold register of TX rings to N, where N >= 0. The default value is 0.

- `--txfreet=N`

  Set the transmit free threshold of TX rings to N, where 0 <= N <= value of `--txd`. The default value is 0.

- `--txrst=N`

  Set the transmit RS bit threshold of TX rings to N, where 0 <= N <= value of `--txd`. The default value is 0.

- `--rx-queue-stats-mapping=(port,queue,mapping)[,(port,queue,mapping)]`

  Set the RX queues statistics counters mapping 0 <= mapping <= 15.

- `--tx-queue-stats-mapping=(port,queue,mapping)[,(port,queue,mapping)]`

  Set the TX queues statistics counters mapping 0 <= mapping <= 15.

- `--no-flush-rx`

  Don't flush the RX streams before starting forwarding. Used mainly with the PCAP PMD.

- `--txpkts=X[,Y]`

  Set TX segment sizes or total packet length. Valid for `tx-only` and `flowgen` forwarding modes.

- `--disable-link-check`

  Disable check on link status when starting/stopping ports.

- `--no-lsc-interrupt`

  Disable LSC interrupts for all ports, even those supporting it.

- `--no-rmv-interrupt`

  Disable RMV interrupts for all ports, even those supporting it.

- `--bitrate-stats=N`

  Set the logical core N to perform bitrate calculation.

- `--print-event <unknown|intr_lsc|queue_state|intr_reset|vf_mbox|macsec|intr_r`

  Enable printing the occurrence of the designated event. Using all will enable all of them.

- `--mask-event <unknown|intr_lsc|queue_state|intr_reset|vf_mbox|macsec|intr_rm`

  Disable printing the occurrence of the designated event. Using all will disable all of them.

- `--flow-isolate-all`

  Providing this parameter requests flow API isolated mode on all ports at initialization time. It ensures all traffic is received through the configured flow rules only (see flow command).

  Ports that do not support this mode are automatically discarded.

- `--tx-offloads=0xXXXXXXXX`

  Set the hexadecimal bitmask of TX queue offloads. The default value is 0.

- `--hot-plug`

  Enable device event monitor machenism for hotplug.

- `--vxlan-gpe-port=N`

  Set the UDP port number of tunnel VXLAN-GPE to N. The default value is 4790.

- `--mlockall`

  Enable locking all memory.

- `--no-mlockall`

  Disable locking all memory.

CHAPTER

FOUR

TESTPMD RUNTIME FUNCTIONS

Where the testpmd application is started in interactive mode, (`-i|--interactive`), it displays a prompt that can be used to start and stop forwarding, configure the application, display statistics (including the extended NIC statistics aka xstats) , set the Flow Director and other tasks:

```
testpmd>
```

The testpmd prompt has some, limited, readline support. Common bash command-line functions such as `Ctrl+a` and `Ctrl+e` to go to the start and end of the prompt line are supported as well as access to the command history via the up-arrow.

There is also support for tab completion. If you type a partial command and hit `<TAB>` you get a list of the available completions:

```
testpmd> show port <TAB>

    info [Mul-choice STRING]: show|clear port info|stats|xstats|fdir|stat_qmap|dcb_tc|cap X
    info [Mul-choice STRING]: show|clear port info|stats|xstats|fdir|stat_qmap|dcb_tc|cap all
    stats [Mul-choice STRING]: show|clear port info|stats|xstats|fdir|stat_qmap|dcb_tc|cap X
    stats [Mul-choice STRING]: show|clear port info|stats|xstats|fdir|stat_qmap|dcb_tc|cap all
    ...
```

Note: Some examples in this document are too long to fit on one line are are shown wrapped at "\" for display purposes:

```
testpmd> set flow_ctrl rx (on|off) tx (on|off) (high_water) (low_water) \
         (pause_time) (send_xon) (port_id)
```

In the real `testpmd>` prompt these commands should be on a single line.

4.1 Help Functions

The testpmd has on-line help for the functions that are available at runtime. These are divided into sections and can be accessed using help, help section or help all:

```
testpmd> help

    help control    : Start and stop forwarding.
    help display    : Displaying port, stats and config information.
    help config     : Configuration information.
    help ports      : Configuring ports.
    help registers  : Reading and setting port registers.
    help filters    : Filters configuration help.
    help all        : All of the above sections.
```

4.2 Command File Functions

To facilitate loading large number of commands or to avoid cutting and pasting where not practical or possible testpmd supports alternative methods for executing commands.

- If started with the `--cmdline-file=FILENAME` command line argument testpmd will execute all CLI commands contained within the file immediately before starting packet forwarding or entering interactive mode.

```
./testpmd -n4 -r2 ... -- -i --cmdline-file=/home/ubuntu/flow-create-commands.txt
Interactive-mode selected
CLI commands to be read from /home/ubuntu/flow-create-commands.txt
Configuring Port 0 (socket 0)
Port 0: 7C:FE:90:CB:74:CE
Configuring Port 1 (socket 0)
Port 1: 7C:FE:90:CB:74:CA
Checking link statuses...
Port 0 Link Up - speed 10000 Mbps - full-duplex
Port 1 Link Up - speed 10000 Mbps - full-duplex
Done
Flow rule #0 created
Flow rule #1 created
...
...
Flow rule #498 created
Flow rule #499 created
Read all CLI commands from /home/ubuntu/flow-create-commands.txt
testpmd>
```

- At run-time additional commands can be loaded in bulk by invoking the `load FILENAME` command.

```
testpmd> load /home/ubuntu/flow-create-commands.txt
Flow rule #0 created
Flow rule #1 created
...
...
Flow rule #498 created
Flow rule #499 created
Read all CLI commands from /home/ubuntu/flow-create-commands.txt
testpmd>
```

In all cases output from any included command will be displayed as standard output. Execution will continue until the end of the file is reached regardless of whether any errors occur. The end user must examine the output to determine if any failures occurred.

4.3 Control Functions

4.3.1 start

Start packet forwarding with current configuration:

```
testpmd> start
```

4.3.2 start tx_first

Start packet forwarding with current configuration after sending specified number of bursts of packets:

```
testpmd> start tx_first ("""|burst_num)
```

The default burst number is 1 when `burst_num` not presented.

4.3.3 stop

Stop packet forwarding, and display accumulated statistics:

```
testpmd> stop
```

4.3.4 quit

Quit to prompt:

```
testpmd> quit
```

4.4 Display Functions

The functions in the following sections are used to display information about the testpmd configuration or the NIC status.

4.4.1 show port

Display information for a given port or all ports:

```
testpmd> show port (info|stats|xstats|fdir|stat_qmap|dcb_tc|cap) (port_id|all)
```

The available information categories are:

- `info`: General port information such as MAC address.
- `stats`: RX/TX statistics.
- `xstats`: RX/TX extended NIC statistics.
- `fdir`: Flow Director information and statistics.
- `stat_qmap`: Queue statistics mapping.
- `dcb_tc`: DCB information such as TC mapping.
- `cap`: Supported offload capabilities.

For example:

```
testpmd> show port info 0

********************* Infos for port 0 *********************

MAC address: XX:XX:XX:XX:XX:XX
Connect to socket: 0
memory allocation on the socket: 0
Link status: up
```

```
Link speed: 40000 Mbps
Link duplex: full-duplex
Promiscuous mode: enabled
Allmulticast mode: disabled
Maximum number of MAC addresses: 64
Maximum number of MAC addresses of hash filtering: 0
VLAN offload:
    strip on
    filter on
    qinq(extend) off
Redirection table size: 512
Supported flow types:
  ipv4-frag
  ipv4-tcp
  ipv4-udp
  ipv4-sctp
  ipv4-other
  ipv6-frag
  ipv6-tcp
  ipv6-udp
  ipv6-sctp
  ipv6-other
  l2_payload
  port
  vxlan
  geneve
  nvgre
```

4.4.2 show port rss reta

Display the rss redirection table entry indicated by masks on port X:

```
testpmd> show port (port_id) rss reta (size) (mask0, mask1...)
```

size is used to indicate the hardware supported reta size

4.4.3 show port rss-hash

Display the RSS hash functions and RSS hash key of a port:

```
testpmd> show port (port_id) rss-hash [key]
```

4.4.4 clear port

Clear the port statistics for a given port or for all ports:

```
testpmd> clear port (info|stats|xstats|fdir|stat_qmap) (port_id|all)
```

For example:

```
testpmd> clear port stats all
```

4.4.5 show (rxq|txq)

Display information for a given port's RX/TX queue:

```
testpmd> show (rxq|txq) info (port_id) (queue_id)
```

4.4.6 show config

Displays the configuration of the application. The configuration comes from the command-line, the runtime or the application defaults:

```
testpmd> show config (rxtx|cores|fwd|txpkts)
```

The available information categories are:

- `rxtx`: RX/TX configuration items.
- `cores`: List of forwarding cores.
- `fwd`: Packet forwarding configuration.
- `txpkts`: Packets to TX configuration.

For example:

```
testpmd> show config rxtx

io packet forwarding - CRC stripping disabled - packets/burst=16
nb forwarding cores=2 - nb forwarding ports=1
RX queues=1 - RX desc=128 - RX free threshold=0
RX threshold registers: pthresh=8 hthresh=8 wthresh=4
TX queues=1 - TX desc=512 - TX free threshold=0
TX threshold registers: pthresh=36 hthresh=0 wthresh=0
TX RS bit threshold=0 - TXQ flags=0x0
```

4.4.7 set fwd

Set the packet forwarding mode:

```
testpmd> set fwd (io|mac|macswap|flowgen| \
                  rxonly|txonly|csum|icmpecho) (""|retry)
```

`retry` can be specified for forwarding engines except `rx_only`.

The available information categories are:

- `io`: Forwards packets "as-is" in I/O mode. This is the fastest possible forwarding operation as it does not access packets data. This is the default mode.
- `mac`: Changes the source and the destination Ethernet addresses of packets before forwarding them. Default application behaviour is to set source Ethernet address to that of the transmitting interface, and destination address to a dummy value (set during init). The user may specify a target destination Ethernet address via the 'eth-peer' or 'eth-peer-configfile' command-line options. It is not currently possible to specify a specific source Ethernet address.
- `macswap`: MAC swap forwarding mode. Swaps the source and the destination Ethernet addresses of packets before forwarding them.
- `flowgen`: Multi-flow generation mode. Originates a number of flows (with varying destination IP addresses), and terminate receive traffic.
- `rxonly`: Receives packets but doesn't transmit them.
- `txonly`: Generates and transmits packets without receiving any.
- `csum`: Changes the checksum field with hardware or software methods depending on the offload flags on the packet.

- `icmpecho`: Receives a burst of packets, lookup for ICMP echo requests and, if any, send back ICMP echo replies.
- `ieee1588`: Demonstrate L2 IEEE1588 V2 PTP timestamping for RX and TX. Requires `CONFIG_RTE_LIBRTE_IEEE1588=y`.
- `softnic`: Demonstrates the softnic forwarding operation. In this mode, packet forwarding is similar to I/O mode except for the fact that packets are loopback to the softnic ports only. Therefore, portmask parameter should be set to softnic port only. The various software based custom NIC pipelines specified through the softnic firmware (DPDK packet framework script) can be tested in this mode. Furthermore, it allows to build 5-level hierarchical QoS scheduler as a default option that can be enabled through CLI once testpmd application is initialised. The user can modify the default scheduler hierarchy or can specify the new QoS Scheduler hierarchy through CLI. Requires `CONFIG_RTE_LIBRTE_PMD_SOFTNIC=y`.

Example:

```
testpmd> set fwd rxonly

Set rxonly packet forwarding mode
```

4.4.8 read rxd

Display an RX descriptor for a port RX queue:

```
testpmd> read rxd (port_id) (queue_id) (rxd_id)
```

For example:

```
testpmd> read rxd 0 0 4
     0x0000000B - 0x001D0180 / 0x0000000B - 0x001D0180
```

4.4.9 read txd

Display a TX descriptor for a port TX queue:

```
testpmd> read txd (port_id) (queue_id) (txd_id)
```

For example:

```
testpmd> read txd 0 0 4
     0x00000001 - 0x24C3C440 / 0x000F0000 - 0x2330003C
```

4.4.10 ddp get list

Get loaded dynamic device personalization (DDP) package info list:

```
testpmd> ddp get list (port_id)
```

4.4.11 ddp get info

Display information about dynamic device personalization (DDP) profile:

```
testpmd> ddp get info (profile_path)
```

4.4.12 show vf stats

Display VF statistics:

```
testpmd> show vf stats (port_id) (vf_id)
```

4.4.13 clear vf stats

Reset VF statistics:

```
testpmd> clear vf stats (port_id) (vf_id)
```

4.4.14 show port pctype mapping

List all items from the pctype mapping table:

```
testpmd> show port (port_id) pctype mapping
```

4.4.15 show rx offloading capabilities

List all per queue and per port Rx offloading capabilities of a port:

```
testpmd> show port (port_id) rx_offload capabilities
```

4.4.16 show rx offloading configuration

List port level and all queue level Rx offloading configuration:

```
testpmd> show port (port_id) rx_offload configuration
```

4.4.17 show tx offloading capabilities

List all per queue and per port Tx offloading capabilities of a port:

```
testpmd> show port (port_id) tx_offload capabilities
```

4.4.18 show tx offloading configuration

List port level and all queue level Tx offloading configuration:

```
testpmd> show port (port_id) tx_offload configuration
```

4.5 Configuration Functions

The testpmd application can be configured from the runtime as well as from the command-line.

This section details the available configuration functions that are available.

Note: Configuration changes only become active when forwarding is started/restarted.

4.5.1 set default

Reset forwarding to the default configuration:

```
testpmd> set default
```

4.5.2 set verbose

Set the debug verbosity level:

```
testpmd> set verbose (level)
```

Currently the only available levels are 0 (silent except for error) and 1 (fully verbose).

4.5.3 set log

Set the log level for a log type:

```
testpmd> set log global|(type) (level)
```

Where:

- `type` is the log name.
- `level` is the log level.

For example, to change the global log level:: testpmd> set log global (level)

Regexes can also be used for type. To change log level of user1, user2 and user3::
testpmd> set log user[1-3] (level)

4.5.4 set nbport

Set the number of ports used by the application:

set nbport (num)

This is equivalent to the `--nb-ports` command-line option.

4.5.5 set nbcore

Set the number of cores used by the application:

```
testpmd> set nbcore (num)
```

This is equivalent to the `--nb-cores` command-line option.

Note: The number of cores used must not be greater than number of ports used multiplied by the number of queues per port.

4.5.6 set coremask

Set the forwarding cores hexadecimal mask:

```
testpmd> set coremask (mask)
```

This is equivalent to the `--coremask` command-line option.

Note: The master lcore is reserved for command line parsing only and cannot be masked on for packet forwarding.

4.5.7 set portmask

Set the forwarding ports hexadecimal mask:

```
testpmd> set portmask (mask)
```

This is equivalent to the `--portmask` command-line option.

4.5.8 set burst

Set number of packets per burst:

```
testpmd> set burst (num)
```

This is equivalent to the `--burst command-line` option.

When retry is enabled, the transmit delay time and number of retries can also be set:

```
testpmd> set burst tx delay (microseconds) retry (num)
```

4.5.9 set txpkts

Set the length of each segment of the TX-ONLY packets or length of packet for FLOWGEN mode:

```
testpmd> set txpkts (x[,y]*)
```

Where x[,y]* represents a CSV list of values, without white space.

4.5.10 set txsplit

Set the split policy for the TX packets, applicable for TX-ONLY and CSUM forwarding modes:

```
testpmd> set txsplit (off|on|rand)
```

Where:

- `off` disable packet copy & split for CSUM mode.
- `on` split outgoing packet into multiple segments. Size of each segment and number of segments per packet is determined by `set txpkts` command (see above).
- `rand` same as 'on', but number of segments per each packet is a random value between 1 and total number of segments.

4.5.11 set corelist

Set the list of forwarding cores:

```
testpmd> set corelist (x[,y]*)
```

For example, to change the forwarding cores:

```
testpmd> set corelist 3,1
testpmd> show config fwd

io packet forwarding - ports=2 - cores=2 - streams=2 - NUMA support disabled
Logical Core 3 (socket 0) forwards packets on 1 streams:
RX P=0/Q=0 (socket 0) -> TX P=1/Q=0 (socket 0) peer=02:00:00:00:00:01
Logical Core 1 (socket 0) forwards packets on 1 streams:
RX P=1/Q=0 (socket 0) -> TX P=0/Q=0 (socket 0) peer=02:00:00:00:00:00
```

Note: The cores are used in the same order as specified on the command line.

4.5.12 set portlist

Set the list of forwarding ports:

```
testpmd> set portlist (x[,y]*)
```

For example, to change the port forwarding:

```
testpmd> set portlist 0,2,1,3
testpmd> show config fwd

io packet forwarding - ports=4 - cores=1 - streams=4
Logical Core 3 (socket 0) forwards packets on 4 streams:
RX P=0/Q=0 (socket 0) -> TX P=2/Q=0 (socket 0) peer=02:00:00:00:00:01
RX P=2/Q=0 (socket 0) -> TX P=0/Q=0 (socket 0) peer=02:00:00:00:00:00
RX P=1/Q=0 (socket 0) -> TX P=3/Q=0 (socket 0) peer=02:00:00:00:00:03
RX P=3/Q=0 (socket 0) -> TX P=1/Q=0 (socket 0) peer=02:00:00:00:00:02
```

4.5.13 set tx loopback

Enable/disable tx loopback:

```
testpmd> set tx loopback (port_id) (on|off)
```

4.5.14 set drop enable

set drop enable bit for all queues:

```
testpmd> set all queues drop (port_id) (on|off)
```

4.5.15 set split drop enable (for VF)

set split drop enable bit for VF from PF:

```
testpmd> set vf split drop (port_id) (vf_id) (on|off)
```

4.5.16 set mac antispoof (for VF)

Set mac antispoof for a VF from the PF:

```
testpmd> set vf mac antispoof  (port_id) (vf_id) (on|off)
```

4.5.17 set macsec offload

Enable/disable MACsec offload:

```
testpmd> set macsec offload (port_id) on encrypt (on|off) replay-protect (on|off)
testpmd> set macsec offload (port_id) off
```

4.5.18 set macsec sc

Configure MACsec secure connection (SC):

```
testpmd> set macsec sc (tx|rx) (port_id) (mac) (pi)
```

Note: The pi argument is ignored for tx. Check the NIC Datasheet for hardware limits.

4.5.19 set macsec sa

Configure MACsec secure association (SA):

```
testpmd> set macsec sa (tx|rx) (port_id) (idx) (an) (pn) (key)
```

Note: The IDX value must be 0 or 1. Check the NIC Datasheet for hardware limits.

4.5.20 set broadcast mode (for VF)

Set broadcast mode for a VF from the PF:

```
testpmd> set vf broadcast (port_id) (vf_id) (on|off)
```

4.5.21 vlan set strip

Set the VLAN strip on a port:

```
testpmd> vlan set strip (on|off) (port_id)
```

4.5.22 vlan set stripq

Set the VLAN strip for a queue on a port:

```
testpmd> vlan set stripq (on|off) (port_id,queue_id)
```

4.5.23 vlan set stripq (for VF)

Set VLAN strip for all queues in a pool for a VF from the PF:

```
testpmd> set vf vlan stripq (port_id) (vf_id) (on|off)
```

4.5.24 vlan set insert (for VF)

Set VLAN insert for a VF from the PF:

```
testpmd> set vf vlan insert (port_id) (vf_id) (vlan_id)
```

4.5.25 vlan set tag (for VF)

Set VLAN tag for a VF from the PF:

```
testpmd> set vf vlan tag (port_id) (vf_id) (on|off)
```

4.5.26 vlan set antispoof (for VF)

Set VLAN antispoof for a VF from the PF:

```
testpmd> set vf vlan antispoof (port_id) (vf_id) (on|off)
```

4.5.27 vlan set filter

Set the VLAN filter on a port:

```
testpmd> vlan set filter (on|off) (port_id)
```

4.5.28 vlan set qinq

Set the VLAN QinQ (extended queue in queue) on for a port:

```
testpmd> vlan set qinq (on|off) (port_id)
```

4.5.29 vlan set tpid

Set the inner or outer VLAN TPID for packet filtering on a port:

```
testpmd> vlan set (inner|outer) tpid (value) (port_id)
```

Note: TPID value must be a 16-bit number (value <= 65536).

4.5.30 rx_vlan add

Add a VLAN ID, or all identifiers, to the set of VLAN identifiers filtered by port ID:

```
testpmd> rx_vlan add (vlan_id|all) (port_id)
```

Note: VLAN filter must be set on that port. VLAN ID < 4096. Depending on the NIC used, number of vlan_ids may be limited to the maximum entries in VFTA table. This is important if enabling all vlan_ids.

4.5.31 rx_vlan rm

Remove a VLAN ID, or all identifiers, from the set of VLAN identifiers filtered by port ID:

```
testpmd> rx_vlan rm (vlan_id|all) (port_id)
```

4.5.32 rx_vlan add (for VF)

Add a VLAN ID, to the set of VLAN identifiers filtered for VF(s) for port ID:

```
testpmd> rx_vlan add (vlan_id) port (port_id) vf (vf_mask)
```

4.5.33 rx_vlan rm (for VF)

Remove a VLAN ID, from the set of VLAN identifiers filtered for VF(s) for port ID:

```
testpmd> rx_vlan rm (vlan_id) port (port_id) vf (vf_mask)
```

4.5.34 tunnel_filter add

Add a tunnel filter on a port:

```
testpmd> tunnel_filter add (port_id) (outer_mac) (inner_mac) (ip_addr) \
         (inner_vlan) (vxlan|nvgre|ipingre) (imac-ivlan|imac-ivlan-tenid|\
         imac-tenid|imac|omac-imac-tenid|oip|iip) (tenant_id) (queue_id)
```

The available information categories are:

- `vxlan`: Set tunnel type as VXLAN.
- `nvgre`: Set tunnel type as NVGRE.
- `ipingre`: Set tunnel type as IP-in-GRE.
- `imac-ivlan`: Set filter type as Inner MAC and VLAN.
- `imac-ivlan-tenid`: Set filter type as Inner MAC, VLAN and tenant ID.
- `imac-tenid`: Set filter type as Inner MAC and tenant ID.
- `imac`: Set filter type as Inner MAC.
- `omac-imac-tenid`: Set filter type as Outer MAC, Inner MAC and tenant ID.
- `oip`: Set filter type as Outer IP.
- `iip`: Set filter type as Inner IP.

Example:

```
testpmd> tunnel_filter add 0 68:05:CA:28:09:82 00:00:00:00:00:00 \
         192.168.2.2 0 ipingre oip 1 1

Set an IP-in-GRE tunnel on port 0, and the filter type is Outer IP.
```

4.5.35 tunnel_filter remove

Remove a tunnel filter on a port:

```
testpmd> tunnel_filter rm (port_id) (outer_mac) (inner_mac) (ip_addr) \
         (inner_vlan) (vxlan|nvgre|ipingre) (imac-ivlan|imac-ivlan-tenid|\
         imac-tenid|imac|omac-imac-tenid|oip|iip) (tenant_id) (queue_id)
```

4.5.36 rx_vxlan_port add

Add an UDP port for VXLAN packet filter on a port:

```
testpmd> rx_vxlan_port add (udp_port) (port_id)
```

4.5.37 rx_vxlan_port remove

Remove an UDP port for VXLAN packet filter on a port:

```
testpmd> rx_vxlan_port rm (udp_port) (port_id)
```

4.5.38 tx_vlan set

Set hardware insertion of VLAN IDs in packets sent on a port:

```
testpmd> tx_vlan set (port_id) vlan_id[, vlan_id_outer]
```

For example, set a single VLAN ID (5) insertion on port 0:

```
tx_vlan set 0 5
```

Or, set double VLAN ID (inner: 2, outer: 3) insertion on port 1:

```
tx_vlan set 1 2 3
```

4.5.39 tx_vlan set pvid

Set port based hardware insertion of VLAN ID in packets sent on a port:

```
testpmd> tx_vlan set pvid (port_id) (vlan_id) (on|off)
```

4.5.40 tx_vlan reset

Disable hardware insertion of a VLAN header in packets sent on a port:

```
testpmd> tx_vlan reset (port_id)
```

4.5.41 csum set

Select hardware or software calculation of the checksum when transmitting a packet using the `csum` forwarding engine:

```
testpmd> csum set (ip|udp|tcp|sctp|outer-ip) (hw|sw) (port_id)
```

Where:

- `ip|udp|tcp|sctp` always relate to the inner layer.

- `outer-ip` relates to the outer IP layer (only for IPv4) in the case where the packet is recognized as a tunnel packet by the forwarding engine (vxlan, gre and ipip are supported). See also the `csum parse-tunnel` command.

Note: Check the NIC Datasheet for hardware limits.

4.5.42 RSS queue region

Set RSS queue region span on a port:

```
testpmd> set port (port_id) queue-region region_id (value) \
             queue_start_index (value) queue_num (value)
```

Set flowtype mapping on a RSS queue region on a port:

```
testpmd> set port (port_id) queue-region region_id (value) flowtype (value)
```

where:

- For the flowtype(pctype) of packet,the specific index for each type has been defined in file i40e_type.h as enum i40e_filter_pctype.

Set user priority mapping on a RSS queue region on a port:

```
testpmd> set port (port_id) queue-region UP (value) region_id (value)
```

Flush all queue region related configuration on a port:

```
testpmd> set port (port_id) queue-region flush (on|off)
```

where:

- “on”is just an enable function which server for other configuration, it is for all configuration about queue region from up layer, at first will only keep in DPDK softwarestored in driver, only after “flush on”, it commit all configuration to HW. “off” is just clean all configuration about queue region just now, and restore all to DPDK i40e driver default config when start up.

Show all queue region related configuration info on a port:

```
testpmd> show port (port_id) queue-region
```

Note: Queue region only support on PF by now, so these command is only for configuration of queue region on PF port.

4.5.43 csum parse-tunnel

Define how tunneled packets should be handled by the csum forward engine:

```
testpmd> csum parse-tunnel (on|off) (tx_port_id)
```

If enabled, the csum forward engine will try to recognize supported tunnel headers (vxlan, gre, ipip).

If disabled, treat tunnel packets as non-tunneled packets (a inner header is handled as a packet payload).

Note: The port argument is the TX port like in the `csum set` command.

Example:

Consider a packet in packet like the following:

```
eth_out/ipv4_out/udp_out/vxlan/eth_in/ipv4_in/tcp_in
```

- If parse-tunnel is enabled, the `ip|udp|tcp|sctp` parameters of `csum set` command relate to the inner headers (here `ipv4_in` and `tcp_in`), and the `outer-ip parameter` relates to the outer headers (here `ipv4_out`).
- **If parse-tunnel is disabled, the `ip|udp|tcp|sctp` parameters of `csum set`** command relate to the outer headers, here `ipv4_out` and `udp_out`.

4.5.44 csum show

Display tx checksum offload configuration:

```
testpmd> csum show (port_id)
```

4.5.45 tso set

Enable TCP Segmentation Offload (TSO) in the `csum` forwarding engine:

```
testpmd> tso set (segsize) (port_id)
```

Note: Check the NIC datasheet for hardware limits.

4.5.46 tso show

Display the status of TCP Segmentation Offload:

```
testpmd> tso show (port_id)
```

4.5.47 set port - gro

Enable or disable GRO in `csum` forwarding engine:

```
testpmd> set port <port_id> gro on|off
```

If enabled, the csum forwarding engine will perform GRO on the TCP/IPv4 packets received from the given port.

If disabled, packets received from the given port won't be performed GRO. By default, GRO is disabled for all ports.

Note: When enable GRO for a port, TCP/IPv4 packets received from the port will be performed GRO. After GRO, all merged packets have bad checksums, since the GRO library doesn't re-calculate checksums for the merged packets. Therefore, if users want the merged packets to have correct checksums, please select HW IP checksum calculation and HW TCP checksum calculation for the port which the merged packets are transmitted to.

4.5.48 show port - gro

Display GRO configuration for a given port:

```
testpmd> show port <port_id> gro
```

4.5.49 set gro flush

Set the cycle to flush the GROed packets from reassembly tables:

```
testpmd> set gro flush <cycles>
```

When enable GRO, the csum forwarding engine performs GRO on received packets, and the GROed packets are stored in reassembly tables. Users can use this command to determine when the GROed packets are flushed from the reassembly tables.

The `cycles` is measured in GRO operation times. The csum forwarding engine flushes the GROed packets from the tables every `cycles` GRO operations.

By default, the value of `cycles` is 1, which means flush GROed packets from the reassembly tables as soon as one GRO operation finishes. The value of `cycles` should be in the range of 1 to `GRO_MAX_FLUSH_CYCLES`.

Please note that the large value of `cycles` may cause the poor TCP/IP stack performance. Because the GROed packets are delayed to arrive the stack, thus causing more duplicated ACKs and TCP retransmissions.

4.5.50 set port - gso

Toggle per-port GSO support in `csum` forwarding engine:

```
testpmd> set port <port_id> gso on|off
```

If enabled, the csum forwarding engine will perform GSO on supported IPv4 packets, transmitted on the given port.

If disabled, packets transmitted on the given port will not undergo GSO. By default, GSO is disabled for all ports.

Note: When GSO is enabled on a port, supported IPv4 packets transmitted on that port undergo GSO. Afterwards, the segmented packets are represented by multi-segment mbufs; however, the csum forwarding engine doesn't calculation of checksums for GSO'd segments in SW. As a result, if users want correct checksums in GSO segments, they should enable HW checksum calculation for GSO-enabled ports.

For example, HW checksum calculation for VxLAN GSO'd packets may be enabled by setting the following options in the csum forwarding engine:

testpmd> csum set outer_ip hw <port_id>

testpmd> csum set ip hw <port_id>

testpmd> csum set tcp hw <port_id>

UDP GSO is the same as IP fragmentation, which treats the UDP header as the payload and does not modify it during segmentation. That is, after UDP GSO, only the first output fragment has the original UDP header. Therefore, users need to enable HW IP checksum calculation

and SW UDP checksum calculation for GSO-enabled ports, if they want correct checksums for UDP/IPv4 packets.

4.5.51 set gso segsz

Set the maximum GSO segment size (measured in bytes), which includes the packet header and the packet payload for GSO-enabled ports (global):

```
testpmd> set gso segsz <length>
```

4.5.52 show port - gso

Display the status of Generic Segmentation Offload for a given port:

```
testpmd> show port <port_id> gso
```

4.5.53 mac_addr add

Add an alternative MAC address to a port:

```
testpmd> mac_addr add (port_id) (XX:XX:XX:XX:XX:XX)
```

4.5.54 mac_addr remove

Remove a MAC address from a port:

```
testpmd> mac_addr remove (port_id) (XX:XX:XX:XX:XX:XX)
```

4.5.55 mac_addr add (for VF)

Add an alternative MAC address for a VF to a port:

```
testpmd> mac_add add port (port_id) vf (vf_id) (XX:XX:XX:XX:XX:XX)
```

4.5.56 mac_addr set

Set the default MAC address for a port:

```
testpmd> mac_addr set (port_id) (XX:XX:XX:XX:XX:XX)
```

4.5.57 mac_addr set (for VF)

Set the MAC address for a VF from the PF:

```
testpmd> set vf mac addr (port_id) (vf_id) (XX:XX:XX:XX:XX:XX)
```

4.5.58 set eth-peer

Set the forwarding peer address for certain port:

```
testpmd> set eth-peer (port_id) (perr_addr)
```

This is equivalent to the `--eth-peer` command-line option.

4.5.59 set port-uta

Set the unicast hash filter(s) on/off for a port:

```
testpmd> set port (port_id) uta (XX:XX:XX:XX:XX:XX|all) (on|off)
```

4.5.60 set promisc

Set the promiscuous mode on for a port or for all ports. In promiscuous mode packets are not dropped if they aren't for the specified MAC address:

```
testpmd> set promisc (port_id|all) (on|off)
```

4.5.61 set allmulti

Set the allmulti mode for a port or for all ports:

```
testpmd> set allmulti (port_id|all) (on|off)
```

Same as the ifconfig (8) option. Controls how multicast packets are handled.

4.5.62 set promisc (for VF)

Set the unicast promiscuous mode for a VF from PF. It's supported by Intel i40e NICs now. In promiscuous mode packets are not dropped if they aren't for the specified MAC address:

```
testpmd> set vf promisc (port_id) (vf_id) (on|off)
```

4.5.63 set allmulticast (for VF)

Set the multicast promiscuous mode for a VF from PF. It's supported by Intel i40e NICs now. In promiscuous mode packets are not dropped if they aren't for the specified MAC address:

```
testpmd> set vf allmulti (port_id) (vf_id) (on|off)
```

4.5.64 set tx max bandwidth (for VF)

Set TX max absolute bandwidth (Mbps) for a VF from PF:

```
testpmd> set vf tx max-bandwidth (port_id) (vf_id) (max_bandwidth)
```

4.5.65 set tc tx min bandwidth (for VF)

Set all TCs' TX min relative bandwidth (%) for a VF from PF:

```
testpmd> set vf tc tx min-bandwidth (port_id) (vf_id) (bw1, bw2, ...)
```

4.5.66 set tc tx max bandwidth (for VF)

Set a TC's TX max absolute bandwidth (Mbps) for a VF from PF:

```
testpmd> set vf tc tx max-bandwidth (port_id) (vf_id) (tc_no) (max_bandwidth)
```

4.5.67 set tc strict link priority mode

Set some TCs' strict link priority mode on a physical port:

```
testpmd> set tx strict-link-priority (port_id) (tc_bitmap)
```

4.5.68 set tc tx min bandwidth

Set all TCs' TX min relative bandwidth (%) globally for all PF and VFs:

```
testpmd> set tc tx min-bandwidth (port_id) (bw1, bw2, ...)
```

4.5.69 set flow_ctrl rx

Set the link flow control parameter on a port:

```
testpmd> set flow_ctrl rx (on|off) tx (on|off) (high_water) (low_water) \
         (pause_time) (send_xon) mac_ctrl_frame_fwd (on|off) \
         autoneg (on|off) (port_id)
```

Where:

- `high_water` (integer): High threshold value to trigger XOFF.
- `low_water` (integer): Low threshold value to trigger XON.
- `pause_time` (integer): Pause quota in the Pause frame.
- `send_xon` (0/1): Send XON frame.
- `mac_ctrl_frame_fwd`: Enable receiving MAC control frames.
- `autoneg`: Change the auto-negotiation parameter.

4.5.70 set pfc_ctrl rx

Set the priority flow control parameter on a port:

```
testpmd> set pfc_ctrl rx (on|off) tx (on|off) (high_water) (low_water) \
         (pause_time) (priority) (port_id)
```

Where:

- `high_water` (integer): High threshold value.
- `low_water` (integer): Low threshold value.

- `pause_time` (integer): Pause quota in the Pause frame.
- `priority` (0-7): VLAN User Priority.

4.5.71 set stat_qmap

Set statistics mapping (qmapping 0..15) for RX/TX queue on port:

```
testpmd> set stat_qmap (tx|rx) (port_id) (queue_id) (qmapping)
```

For example, to set rx queue 2 on port 0 to mapping 5:

```
testpmd>set stat_qmap rx 0 2 5
```

4.5.72 set xstats-hide-zero

Set the option to hide zero values for xstats display:

```
testpmd> set xstats-hide-zero on|off
```

Note: By default, the zero values are displayed for xstats.

4.5.73 set port - rx/tx (for VF)

Set VF receive/transmit from a port:

```
testpmd> set port (port_id) vf (vf_id) (rx|tx) (on|off)
```

4.5.74 set port - mac address filter (for VF)

Add/Remove unicast or multicast MAC addr filter for a VF:

```
testpmd> set port (port_id) vf (vf_id) (mac_addr) \
         (exact-mac|exact-mac-vlan|hashmac|hashmac-vlan) (on|off)
```

4.5.75 set port - rx mode(for VF)

Set the VF receive mode of a port:

```
testpmd> set port (port_id) vf (vf_id) \
         rxmode (AUPE|ROPE|BAM|MPE) (on|off)
```

The available receive modes are:

- `AUPE`: Accepts untagged VLAN.
- `ROPE`: Accepts unicast hash.
- `BAM`: Accepts broadcast packets.
- `MPE`: Accepts all multicast packets.

4.5.76 set port - tx_rate (for Queue)

Set TX rate limitation for a queue on a port:

```
testpmd> set port (port_id) queue (queue_id) rate (rate_value)
```

4.5.77 set port - tx_rate (for VF)

Set TX rate limitation for queues in VF on a port:

```
testpmd> set port (port_id) vf (vf_id) rate (rate_value) queue_mask (queue_mask)
```

4.5.78 set port - mirror rule

Set pool or vlan type mirror rule for a port:

```
testpmd> set port (port_id) mirror-rule (rule_id) \
         (pool-mirror-up|pool-mirror-down|vlan-mirror) \
         (poolmask|vlanid[,vlanid]*) dst-pool (pool_id) (on|off)
```

Set link mirror rule for a port:

```
testpmd> set port (port_id) mirror-rule (rule_id) \
        (uplink-mirror|downlink-mirror) dst-pool (pool_id) (on|off)
```

For example to enable mirror traffic with vlan 0,1 to pool 0:

```
set port 0 mirror-rule 0 vlan-mirror 0,1 dst-pool 0 on
```

4.5.79 reset port - mirror rule

Reset a mirror rule for a port:

```
testpmd> reset port (port_id) mirror-rule (rule_id)
```

4.5.80 set flush_rx

Set the flush on RX streams before forwarding. The default is flush `on`. Mainly used with PCAP drivers to turn off the default behavior of flushing the first 512 packets on RX streams:

```
testpmd> set flush_rx off
```

4.5.81 set bypass mode

Set the bypass mode for the lowest port on bypass enabled NIC:

```
testpmd> set bypass mode (normal|bypass|isolate) (port_id)
```

4.5.82 set bypass event

Set the event required to initiate specified bypass mode for the lowest port on a bypass enabled:

```
testpmd> set bypass event (timeout|os_on|os_off|power_on|power_off) \
         mode (normal|bypass|isolate) (port_id)
```

Where:

- `timeout`: Enable bypass after watchdog timeout.
- `os_on`: Enable bypass when OS/board is powered on.
- `os_off`: Enable bypass when OS/board is powered off.
- `power_on`: Enable bypass when power supply is turned on.
- `power_off`: Enable bypass when power supply is turned off.

4.5.83 set bypass timeout

Set the bypass watchdog timeout to `n` seconds where 0 = instant:

```
testpmd> set bypass timeout (0|1.5|2|3|4|8|16|32)
```

4.5.84 show bypass config

Show the bypass configuration for a bypass enabled NIC using the lowest port on the NIC:

```
testpmd> show bypass config (port_id)
```

4.5.85 set link up

Set link up for a port:

```
testpmd> set link-up port (port id)
```

4.5.86 set link down

Set link down for a port:

```
testpmd> set link-down port (port id)
```

4.5.87 E-tag set

Enable E-tag insertion for a VF on a port:

```
testpmd> E-tag set insertion on port-tag-id (value) port (port_id) vf (vf_id)
```

Disable E-tag insertion for a VF on a port:

```
testpmd> E-tag set insertion off port (port_id) vf (vf_id)
```

Enable/disable E-tag stripping on a port:

```
testpmd> E-tag set stripping (on|off) port (port_id)
```

Enable/disable E-tag based forwarding on a port:

```
testpmd> E-tag set forwarding (on|off) port (port_id)
```

Add an E-tag forwarding filter on a port:

```
testpmd> E-tag set filter add e-tag-id (value) dst-pool (pool_id) port (port_id)
```

Delete an E-tag forwarding filter on a port:: testpmd> E-tag set filter del e-tag-id (value) port (port_id)

4.5.88 ddp add

Load a dynamic device personalization (DDP) profile and store backup profile:

```
testpmd> ddp add (port_id) (profile_path[,backup_profile_path])
```

4.5.89 ddp del

Delete a dynamic device personalization profile and restore backup profile:

```
testpmd> ddp del (port_id) (backup_profile_path)
```

4.5.90 ptype mapping

List all items from the ptype mapping table:

```
testpmd> ptype mapping get (port_id) (valid_only)
```

Where:

- `valid_only`: A flag indicates if only list valid items(=1) or all itemss(=0).

Replace a specific or a group of software defined ptype with a new one:

```
testpmd> ptype mapping replace  (port_id) (target) (mask) (pkt_type)
```

where:

- `target`: A specific software ptype or a mask to represent a group of software ptypes.
- `mask`: A flag indicate if "target" is a specific software ptype(=0) or a ptype mask(=1).
- `pkt_type`: The new software ptype to replace the old ones.

Update hardware defined ptype to software defined packet type mapping table:

```
testpmd> ptype mapping update (port_id) (hw_ptype) (sw_ptype)
```

where:

- `hw_ptype`: hardware ptype as the index of the ptype mapping table.
- `sw_ptype`: software ptype as the value of the ptype mapping table.

Reset ptype mapping table:

```
testpmd> ptype mapping reset (port_id)
```

4.5.91 config per port Rx offloading

Enable or disable a per port Rx offloading on all Rx queues of a port:

```
testpmd> port config (port_id) rx_offload (offloading) on|off
```

- **offloading: can be any of these offloading capability:** vlan_strip, ipv4_cksum, udp_cksum, tcp_cksum, tcp_lro, qinq_strip, outer_ipv4_cksum, macsec_strip, header_split, vlan_filter, vlan_extend, jumbo_frame, crc_strip, scatter, timestamp, security, keep_crc

This command should be run when the port is stopped, or else it will fail.

4.5.92 config per queue Rx offloading

Enable or disable a per queue Rx offloading only on a specific Rx queue:

```
testpmd> port (port_id) rxq (queue_id) rx_offload (offloading) on|off
```

- **offloading: can be any of these offloading capability:** vlan_strip, ipv4_cksum, udp_cksum, tcp_cksum, tcp_lro, qinq_strip, outer_ipv4_cksum, macsec_strip, header_split, vlan_filter, vlan_extend, jumbo_frame, crc_strip, scatter, timestamp, security, keep_crc

This command should be run when the port is stopped, or else it will fail.

4.5.93 config per port Tx offloading

Enable or disable a per port Tx offloading on all Tx queues of a port:

```
testpmd> port config (port_id) tx_offload (offloading) on|off
```

- **offloading: can be any of these offloading capability:** vlan_insert, ipv4_cksum, udp_cksum, tcp_cksum, sctp_cksum, tcp_tso, udp_tso, outer_ipv4_cksum, qinq_insert, vxlan_tnl_tso, gre_tnl_tso, ipip_tnl_tso, geneve_tnl_tso, macsec_insert, mt_lockfree, multi_segs, mbuf_fast_free, security

This command should be run when the port is stopped, or else it will fail.

4.5.94 config per queue Tx offloading

Enable or disable a per queue Tx offloading only on a specific Tx queue:

```
testpmd> port (port_id) txq (queue_id) tx_offload (offloading) on|off
```

- **offloading: can be any of these offloading capability:** vlan_insert, ipv4_cksum, udp_cksum, tcp_cksum, sctp_cksum, tcp_tso, udp_tso, outer_ipv4_cksum, qinq_insert, vxlan_tnl_tso, gre_tnl_tso, ipip_tnl_tso, geneve_tnl_tso, macsec_insert, mt_lockfree, multi_segs, mbuf_fast_free, security

This command should be run when the port is stopped, or else it will fail.

4.5.95 Config VXLAN Encap outer layers

Configure the outer layer to encapsulate a packet inside a VXLAN tunnel:

```
set vxlan ip-version (ipv4|ipv6) vni (vni) udp-src (udp-src) \
udp-dst (udp-dst) ip-src (ip-src) ip-dst (ip-dst) eth-src (eth-src) \
eth-dst (eth-dst)

set vxlan-with-vlan ip-version (ipv4|ipv6) vni (vni) udp-src (udp-src) \
```

```
udp-dst (udp-dst) ip-src (ip-src) ip-dst (ip-dst) vlan-tci (vlan-tci) \
eth-src (eth-src) eth-dst (eth-dst)
```

Those command will set an internal configuration inside testpmd, any following flow rule using the action vxlan_encap will use the last configuration set. To have a different encapsulation header, one of those commands must be called before the flow rule creation.

4.5.96 Config NVGRE Encap outer layers

Configure the outer layer to encapsulate a packet inside a NVGRE tunnel:

```
set nvgre ip-version (ipv4|ipv6) tni (tni) ip-src (ip-src) ip-dst (ip-dst) \
       eth-src (eth-src) eth-dst (eth-dst)
set nvgre-with-vlan ip-version (ipv4|ipv6) tni (tni) ip-src (ip-src) \
       ip-dst (ip-dst) vlan-tci (vlan-tci) eth-src (eth-src) eth-dst (eth-dst)
```

Those command will set an internal configuration inside testpmd, any following flow rule using the action nvgre_encap will use the last configuration set. To have a different encapsulation header, one of those commands must be called before the flow rule creation.

4.6 Port Functions

The following sections show functions for configuring ports.

Note: Port configuration changes only become active when forwarding is started/restarted.

4.6.1 port attach

Attach a port specified by pci address or virtual device args:

```
testpmd> port attach (identifier)
```

To attach a new pci device, the device should be recognized by kernel first. Then it should be moved under DPDK management. Finally the port can be attached to testpmd.

For example, to move a pci device using ixgbe under DPDK management:

```
# Check the status of the available devices.
./usertools/dpdk-devbind.py --status

Network devices using DPDK-compatible driver
============================================
<none>

Network devices using kernel driver
===================================
0000:0a:00.0 '82599ES 10-Gigabit' if=eth2 drv=ixgbe unused=


# Bind the device to igb_uio.
sudo ./usertools/dpdk-devbind.py -b igb_uio 0000:0a:00.0


# Recheck the status of the devices.
./usertools/dpdk-devbind.py --status
Network devices using DPDK-compatible driver
```

```
============================================
0000:0a:00.0 '82599ES 10-Gigabit' drv=igb_uio unused=
```

To attach a port created by virtual device, above steps are not needed.

For example, to attach a port whose pci address is 0000:0a:00.0.

```
testpmd> port attach 0000:0a:00.0
Attaching a new port...
EAL: PCI device 0000:0a:00.0 on NUMA socket -1
EAL:   probe driver: 8086:10fb rte_ixgbe_pmd
EAL:   PCI memory mapped at 0x7f83bfa00000
EAL:   PCI memory mapped at 0x7f83bfa80000
PMD: eth_ixgbe_dev_init(): MAC: 2, PHY: 18, SFP+: 5
PMD: eth_ixgbe_dev_init(): port 0 vendorID=0x8086 deviceID=0x10fb
Port 0 is attached. Now total ports is 1
Done
```

For example, to attach a port created by pcap PMD.

```
testpmd> port attach net_pcap0
Attaching a new port...
PMD: Initializing pmd_pcap for net_pcap0
PMD: Creating pcap-backed ethdev on numa socket 0
Port 0 is attached. Now total ports is 1
Done
```

In this case, identifier is `net_pcap0`. This identifier format is the same as `--vdev` format of DPDK applications.

For example, to re-attach a bonded port which has been previously detached, the mode and slave parameters must be given.

```
testpmd> port attach net_bond_0,mode=0,slave=1
Attaching a new port...
EAL: Initializing pmd_bond for net_bond_0
EAL: Create bonded device net_bond_0 on port 0 in mode 0 on socket 0.
Port 0 is attached. Now total ports is 1
Done
```

4.6.2 port detach

Detach a specific port:

```
testpmd> port detach (port_id)
```

Before detaching a port, the port should be stopped and closed.

For example, to detach a pci device port 0.

```
testpmd> port stop 0
Stopping ports...
Done
testpmd> port close 0
Closing ports...
Done

testpmd> port detach 0
Detaching a port...
EAL: PCI device 0000:0a:00.0 on NUMA socket -1
EAL:   remove driver: 8086:10fb rte_ixgbe_pmd
EAL:   PCI memory unmapped at 0x7f83bfa00000
EAL:   PCI memory unmapped at 0x7f83bfa80000
Done
```

For example, to detach a virtual device port 0.

```
testpmd> port stop 0
Stopping ports...
Done
testpmd> port close 0
Closing ports...
Done

testpmd> port detach 0
Detaching a port...
PMD: Closing pcap ethdev on numa socket 0
Port 'net_pcap0' is detached. Now total ports is 0
Done
```

To remove a pci device completely from the system, first detach the port from testpmd. Then the device should be moved under kernel management. Finally the device can be removed using kernel pci hotplug functionality.

For example, to move a pci device under kernel management:

```
sudo ./usertools/dpdk-devbind.py -b ixgbe 0000:0a:00.0

./usertools/dpdk-devbind.py --status

Network devices using DPDK-compatible driver
============================================
<none>

Network devices using kernel driver
===================================
0000:0a:00.0 '82599ES 10-Gigabit' if=eth2 drv=ixgbe unused=igb_uio
```

To remove a port created by a virtual device, above steps are not needed.

4.6.3 port start

Start all ports or a specific port:

```
testpmd> port start (port_id|all)
```

4.6.4 port stop

Stop all ports or a specific port:

```
testpmd> port stop (port_id|all)
```

4.6.5 port close

Close all ports or a specific port:

```
testpmd> port close (port_id|all)
```

4.6.6 port config - queue ring size

Configure a rx/tx queue ring size:

```
testpmd> port (port_id) (rxq|txq) (queue_id) ring_size (value)
```

Only take effect after command that (re-)start the port or command that setup specific queue.

4.6.7 port start/stop queue

Start/stop a rx/tx queue on a specific port:

```
testpmd> port (port_id) (rxq|txq) (queue_id) (start|stop)
```

4.6.8 port setup queue

Setup a rx/tx queue on a specific port:

```
testpmd> port (port_id) (rxq|txq) (queue_id) setup
```

Only take effect when port is started.

4.6.9 port config - speed

Set the speed and duplex mode for all ports or a specific port:

```
testpmd> port config (port_id|all) speed (10|100|1000|10000|25000|40000|50000|100000|auto) \
         duplex (half|full|auto)
```

4.6.10 port config - queues/descriptors

Set number of queues/descriptors for rxq, txq, rxd and txd:

```
testpmd> port config all (rxq|txq|rxd|txd) (value)
```

This is equivalent to the `--rxq`, `--txq`, `--rxd` and `--txd` command-line options.

4.6.11 port config - max-pkt-len

Set the maximum packet length:

```
testpmd> port config all max-pkt-len (value)
```

This is equivalent to the `--max-pkt-len` command-line option.

4.6.12 port config - CRC Strip

Set hardware CRC stripping on or off for all ports:

```
testpmd> port config all crc-strip (on|off)
```

CRC stripping is on by default.

The `off` option is equivalent to the `--disable-crc-strip` command-line option.

4.6.13 port config - scatter

Set RX scatter mode on or off for all ports:

```
testpmd> port config all scatter (on|off)
```

RX scatter mode is off by default.

The `on` option is equivalent to the `--enable-scatter` command-line option.

4.6.14 port config - RX Checksum

Set hardware RX checksum offload to on or off for all ports:

```
testpmd> port config all rx-cksum (on|off)
```

Checksum offload is off by default.

The `on` option is equivalent to the `--enable-rx-cksum` command-line option.

4.6.15 port config - VLAN

Set hardware VLAN on or off for all ports:

```
testpmd> port config all hw-vlan (on|off)
```

Hardware VLAN is off by default.

The `on` option is equivalent to the `--enable-hw-vlan` command-line option.

4.6.16 port config - VLAN filter

Set hardware VLAN filter on or off for all ports:

```
testpmd> port config all hw-vlan-filter (on|off)
```

Hardware VLAN filter is off by default.

The `on` option is equivalent to the `--enable-hw-vlan-filter` command-line option.

4.6.17 port config - VLAN strip

Set hardware VLAN strip on or off for all ports:

```
testpmd> port config all hw-vlan-strip (on|off)
```

Hardware VLAN strip is off by default.

The `on` option is equivalent to the `--enable-hw-vlan-strip` command-line option.

4.6.18 port config - VLAN extend

Set hardware VLAN extend on or off for all ports:

```
testpmd> port config all hw-vlan-extend (on|off)
```

Hardware VLAN extend is off by default.

The `on` option is equivalent to the `--enable-hw-vlan-extend` command-line option.

4.6.19 port config - Drop Packets

Set packet drop for packets with no descriptors on or off for all ports:

```
testpmd> port config all drop-en (on|off)
```

Packet dropping for packets with no descriptors is off by default.

The `on` option is equivalent to the `--enable-drop-en` command-line option.

4.6.20 port config - RSS

Set the RSS (Receive Side Scaling) mode on or off:

```
testpmd> port config all rss (all|default|ip|tcp|udp|sctp|ether|port|vxlan|geneve|nvgre|none)
```

RSS is on by default.

The `all` option is equivalent to ip|tcp|udp|sctp|ether. The `default` option enables all supported RSS types reported by device info. The `none` option is equivalent to the `--disable-rss` command-line option.

4.6.21 port config - RSS Reta

Set the RSS (Receive Side Scaling) redirection table:

```
testpmd> port config all rss reta (hash,queue)[,(hash,queue)]
```

4.6.22 port config - DCB

Set the DCB mode for an individual port:

```
testpmd> port config (port_id) dcb vt (on|off) (traffic_class) pfc (on|off)
```

The traffic class should be 4 or 8.

4.6.23 port config - Burst

Set the number of packets per burst:

```
testpmd> port config all burst (value)
```

This is equivalent to the `--burst` command-line option.

4.6.24 port config - Threshold

Set thresholds for TX/RX queues:

```
testpmd> port config all (threshold) (value)
```

Where the threshold type can be:

- `txpt`: Set the prefetch threshold register of the TX rings, 0 <= value <= 255.
- `txht`: Set the host threshold register of the TX rings, 0 <= value <= 255.
- `txwt`: Set the write-back threshold register of the TX rings, 0 <= value <= 255.
- `rxpt`: Set the prefetch threshold register of the RX rings, 0 <= value <= 255.
- `rxht`: Set the host threshold register of the RX rings, 0 <= value <= 255.
- `rxwt`: Set the write-back threshold register of the RX rings, 0 <= value <= 255.
- `txfreet`: Set the transmit free threshold of the TX rings, 0 <= value <= txd.
- `rxfreet`: Set the transmit free threshold of the RX rings, 0 <= value <= rxd.
- `txrst`: Set the transmit RS bit threshold of TX rings, 0 <= value <= txd.

These threshold options are also available from the command-line.

4.6.25 port config - E-tag

Set the value of ether-type for E-tag:

```
testpmd> port config (port_id|all) l2-tunnel E-tag ether-type (value)
```

Enable/disable the E-tag support:

```
testpmd> port config (port_id|all) l2-tunnel E-tag (enable|disable)
```

4.6.26 port config pctype mapping

Reset pctype mapping table:

```
testpmd> port config (port_id) pctype mapping reset
```

Update hardware defined pctype to software defined flow type mapping table:

```
testpmd> port config (port_id) pctype mapping update (pctype_id_0[,pctype_id_1]*) (flow_type_id
```

where:

- `pctype_id_x`: hardware pctype id as index of bit in bitmask value of the pctype mapping table.
- `flow_type_id`: software flow type id as the index of the pctype mapping table.

4.6.27 port config input set

Config RSS/FDIR/FDIR flexible payload input set for some pctype::

testpmd> port config (port_id) pctype (pctype_id) (hash_inset|fdir_inset|fdir_flx_inset) (get|set|clear) field (field_idx)

Clear RSS/FDIR/FDIR flexible payload input set for some pctype::

testpmd> port config (port_id) pctype (pctype_id) (hash_inset|fdir_inset|fdir_flx_inset) clear all

where:

- `pctype_id`: hardware packet classification types.
- `field_idx`: hardware field index.

4.6.28 port config udp_tunnel_port

Add/remove UDP tunnel port for VXLAN/GENEVE tunneling protocols:: testpmd> port config (port_id) udp_tunnel_port add|rm vxlan|geneve (udp_port)

4.7 Link Bonding Functions

The Link Bonding functions make it possible to dynamically create and manage link bonding devices from within testpmd interactive prompt.

4.7.1 create bonded device

Create a new bonding device:

```
testpmd> create bonded device (mode) (socket)
```

For example, to create a bonded device in mode 1 on socket 0:

```
testpmd> create bonded device 1 0
created new bonded device (port X)
```

4.7.2 add bonding slave

Adds Ethernet device to a Link Bonding device:

```
testpmd> add bonding slave (slave id) (port id)
```

For example, to add Ethernet device (port 6) to a Link Bonding device (port 10):

```
testpmd> add bonding slave 6 10
```

4.7.3 remove bonding slave

Removes an Ethernet slave device from a Link Bonding device:

```
testpmd> remove bonding slave (slave id) (port id)
```

For example, to remove Ethernet slave device (port 6) to a Link Bonding device (port 10):

```
testpmd> remove bonding slave 6 10
```

4.7.4 set bonding mode

Set the Link Bonding mode of a Link Bonding device:

```
testpmd> set bonding mode (value) (port id)
```

For example, to set the bonding mode of a Link Bonding device (port 10) to broadcast (mode 3):

```
testpmd> set bonding mode 3 10
```

4.7.5 set bonding primary

Set an Ethernet slave device as the primary device on a Link Bonding device:

```
testpmd> set bonding primary (slave id) (port id)
```

For example, to set the Ethernet slave device (port 6) as the primary port of a Link Bonding device (port 10):

```
testpmd> set bonding primary 6 10
```

4.7.6 set bonding mac

Set the MAC address of a Link Bonding device:

```
testpmd> set bonding mac (port id) (mac)
```

For example, to set the MAC address of a Link Bonding device (port 10) to 00:00:00:00:00:01:

```
testpmd> set bonding mac 10 00:00:00:00:00:01
```

4.7.7 set bonding xmit_balance_policy

Set the transmission policy for a Link Bonding device when it is in Balance XOR mode:

```
testpmd> set bonding xmit_balance_policy (port_id) (l2|l23|l34)
```

For example, set a Link Bonding device (port 10) to use a balance policy of layer 3+4 (IP addresses & UDP ports):

```
testpmd> set bonding xmit_balance_policy 10 l34
```

4.7.8 set bonding mon_period

Set the link status monitoring polling period in milliseconds for a bonding device.

This adds support for PMD slave devices which do not support link status interrupts. When the mon_period is set to a value greater than 0 then all PMD's which do not support link status ISR will be queried every polling interval to check if their link status has changed:

```
testpmd> set bonding mon_period (port_id) (value)
```

For example, to set the link status monitoring polling period of bonded device (port 5) to 150ms:

```
testpmd> set bonding mon_period 5 150
```

4.7.9 set bonding lacp dedicated_queue

Enable dedicated tx/rx queues on bonding devices slaves to handle LACP control plane traffic when in mode 4 (link-aggregration-802.3ad):

```
testpmd> set bonding lacp dedicated_queues (port_id) (enable|disable)
```

4.7.10 set bonding agg_mode

Enable one of the specific aggregators mode when in mode 4 (link-aggregration-802.3ad):

```
testpmd> set bonding agg_mode (port_id) (bandwidth|count|stable)
```

4.7.11 show bonding config

Show the current configuration of a Link Bonding device:

```
testpmd> show bonding config (port id)
```

For example, to show the configuration a Link Bonding device (port 9) with 3 slave devices (1, 3, 4) in balance mode with a transmission policy of layer 2+3:

```
testpmd> show bonding config 9
     Bonding mode: 2
     Balance Xmit Policy: BALANCE_XMIT_POLICY_LAYER23
     Slaves (3): [1 3 4]
     Active Slaves (3): [1 3 4]
     Primary: [3]
```

4.8 Register Functions

The Register Functions can be used to read from and write to registers on the network card referenced by a port number. This is mainly useful for debugging purposes. Reference should be made to the appropriate datasheet for the network card for details on the register addresses and fields that can be accessed.

4.8.1 read reg

Display the value of a port register:

```
testpmd> read reg (port_id) (address)
```

For example, to examine the Flow Director control register (FDIRCTL, 0x0000EE000) on an Intel 82599 10 GbE Controller:

```
testpmd> read reg 0 0xEE00
port 0 PCI register at offset 0xEE00: 0x4A060029 (1241907241)
```

4.8.2 read regfield

Display a port register bit field:

```
testpmd> read regfield (port_id) (address) (bit_x) (bit_y)
```

For example, reading the lowest two bits from the register in the example above:

```
testpmd> read regfield 0 0xEE00 0 1
port 0 PCI register at offset 0xEE00: bits[0, 1]=0x1 (1)
```

4.8.3 read regbit

Display a single port register bit:

```
testpmd> read regbit (port_id) (address) (bit_x)
```

For example, reading the lowest bit from the register in the example above:

```
testpmd> read regbit 0 0xEE00 0
port 0 PCI register at offset 0xEE00: bit 0=1
```

4.8.4 write reg

Set the value of a port register:

```
testpmd> write reg (port_id) (address) (value)
```

For example, to clear a register:

```
testpmd> write reg 0 0xEE00 0x0
port 0 PCI register at offset 0xEE00: 0x00000000 (0)
```

4.8.5 write regfield

Set bit field of a port register:

```
testpmd> write regfield (port_id) (address) (bit_x) (bit_y) (value)
```

For example, writing to the register cleared in the example above:

```
testpmd> write regfield 0 0xEE00 0 1 2
port 0 PCI register at offset 0xEE00: 0x00000002 (2)
```

4.8.6 write regbit

Set single bit value of a port register:

```
testpmd> write regbit (port_id) (address) (bit_x) (value)
```

For example, to set the high bit in the register from the example above:

```
testpmd> write regbit 0 0xEE00 31 1
port 0 PCI register at offset 0xEE00: 0x8000000A (2147483658)
```

4.9 Traffic Metering and Policing

The following section shows functions for configuring traffic metering and policing on the ethernet device through the use of generic ethdev API.

4.9.1 show port traffic management capability

Show traffic metering and policing capability of the port:

```
testpmd> show port meter cap (port_id)
```

4.9.2 add port meter profile (srTCM rfc2967)

Add meter profile (srTCM rfc2697) to the ethernet device:

```
testpmd> add port meter profile srtcm_rfc2697 (port_id) (profile_id) \
(cir) (cbs) (ebs)
```

where:

- `profile_id`: ID for the meter profile.
- `cir`: Committed Information Rate (CIR) (bytes/second).
- `cbs`: Committed Burst Size (CBS) (bytes).
- `ebs`: Excess Burst Size (EBS) (bytes).

4.9.3 add port meter profile (trTCM rfc2968)

Add meter profile (srTCM rfc2698) to the ethernet device:

```
testpmd> add port meter profile trtcm_rfc2698 (port_id) (profile_id) \
(cir) (pir) (cbs) (pbs)
```

where:

- `profile_id`: ID for the meter profile.
- `cir`: Committed information rate (bytes/second).
- `pir`: Peak information rate (bytes/second).
- `cbs`: Committed burst size (bytes).
- `pbs`: Peak burst size (bytes).

4.9.4 add port meter profile (trTCM rfc4115)

Add meter profile (trTCM rfc4115) to the ethernet device:

```
testpmd> add port meter profile trtcm_rfc4115 (port_id) (profile_id) \
(cir) (eir) (cbs) (ebs)
```

where:

- `profile_id`: ID for the meter profile.
- `cir`: Committed information rate (bytes/second).
- `eir`: Excess information rate (bytes/second).
- `cbs`: Committed burst size (bytes).
- `ebs`: Excess burst size (bytes).

4.9.5 delete port meter profile

Delete meter profile from the ethernet device:

```
testpmd> del port meter profile (port_id) (profile_id)
```

4.9.6 create port meter

Create new meter object for the ethernet device:

```
testpmd> create port meter (port_id) (mtr_id) (profile_id) \
(meter_enable) (g_action) (y_action) (r_action) (stats_mask) (shared) \
(use_pre_meter_color) [(dscp_tbl_entry0) (dscp_tbl_entry1)...\
(dscp_tbl_entry63)]
```

where:

- `mtr_id`: meter object ID.
- `profile_id`: ID for the meter profile.
- `meter_enable`: When this parameter has a non-zero value, the meter object gets enabled at the time of creation, otherwise remains disabled.
- `g_action`: Policer action for the packet with green color.
- `y_action`: Policer action for the packet with yellow color.
- `r_action`: Policer action for the packet with red color.
- `stats_mask`: Mask of statistics counter types to be enabled for the meter object.
- `shared`: When this parameter has a non-zero value, the meter object is shared by multiple flows. Otherwise, meter object is used by single flow.
- `use_pre_meter_color`: When this parameter has a non-zero value, the input color for the current meter object is determined by the latest meter object in the same flow. Otherwise, the current meter object uses the *dscp_table* to determine the input color.
- `dscp_tbl_entryx`: DSCP table entry x providing meter providing input color, 0 <= x <= 63.

4.9.7 enable port meter

Enable meter for the ethernet device:

```
testpmd> enable port meter (port_id) (mtr_id)
```

4.9.8 disable port meter

Disable meter for the ethernet device:

```
testpmd> disable port meter (port_id) (mtr_id)
```

4.9.9 delete port meter

Delete meter for the ethernet device:

```
testpmd> del port meter (port_id) (mtr_id)
```

4.9.10 Set port meter profile

Set meter profile for the ethernet device:

```
testpmd> set port meter profile (port_id) (mtr_id) (profile_id)
```

4.9.11 set port meter dscp table

Set meter dscp table for the ethernet device:

```
testpmd> set port meter dscp table (port_id) (mtr_id) [(dscp_tbl_entry0) \
(dscp_tbl_entry1)...(dscp_tbl_entry63)]
```

4.9.12 set port meter policer action

Set meter policer action for the ethernet device:

```
testpmd> set port meter policer action (port_id) (mtr_id) (action_mask) \
(action0) [(action1) (action1)]
```

where:

- `action_mask`: Bit mask indicating which policer actions need to be updated. One or more policer actions can be updated in a single function invocation. To update the policer action associated with color C, bit (1 << C) needs to be set in *action_mask* and element at position C in the *actions* array needs to be valid.
- `actionx`: Policer action for the color x, RTE_MTR_GREEN <= x < RTE_MTR_COLORS

4.9.13 set port meter stats mask

Set meter stats mask for the ethernet device:

```
testpmd> set port meter stats mask (port_id) (mtr_id) (stats_mask)
```

where:

- `stats_mask`: Bit mask indicating statistics counter types to be enabled.

4.9.14 show port meter stats

Show meter stats of the ethernet device:

```
testpmd> show port meter stats (port_id) (mtr_id) (clear)
```

where:

- `clear`: Flag that indicates whether the statistics counters should be cleared (i.e. set to zero) immediately after they have been read or not.

4.10 Traffic Management

The following section shows functions for configuring traffic management on on the ethernet device through the use of generic TM API.

4.10.1 show port traffic management capability

Show traffic management capability of the port:

```
testpmd> show port tm cap (port_id)
```

4.10.2 show port traffic management capability (hierarchy level)

Show traffic management hierarchy level capability of the port:

```
testpmd> show port tm level cap (port_id) (level_id)
```

4.10.3 show port traffic management capability (hierarchy node level)

Show the traffic management hierarchy node capability of the port:

```
testpmd> show port tm node cap (port_id) (node_id)
```

4.10.4 show port traffic management hierarchy node type

Show the port traffic management hierarchy node type:

```
testpmd> show port tm node type (port_id) (node_id)
```

4.10.5 show port traffic management hierarchy node stats

Show the port traffic management hierarchy node statistics:

```
testpmd> show port tm node stats (port_id) (node_id) (clear)
```

where:

- `clear`: When this parameter has a non-zero value, the statistics counters are cleared (i.e. set to zero) immediately after they have been read, otherwise the statistics counters are left untouched.

4.10.6 Add port traffic management private shaper profile

Add the port traffic management private shaper profile:

```
testpmd> add port tm node shaper profile (port_id) (shaper_profile_id) \
(cmit_tb_rate) (cmit_tb_size) (peak_tb_rate) (peak_tb_size) \
(packet_length_adjust)
```

where:

- `shaper_profile id`: Shaper profile ID for the new profile.
- `cmit_tb_rate`: Committed token bucket rate (bytes per second).
- `cmit_tb_size`: Committed token bucket size (bytes).
- `peak_tb_rate`: Peak token bucket rate (bytes per second).
- `peak_tb_size`: Peak token bucket size (bytes).

- `packet_length_adjust`: The value (bytes) to be added to the length of each packet for the purpose of shaping. This parameter value can be used to correct the packet length with the framing overhead bytes that are consumed on the wire.

4.10.7 Delete port traffic management private shaper profile

Delete the port traffic management private shaper:

```
testpmd> del port tm node shaper profile (port_id) (shaper_profile_id)
```

where:

- `shaper_profile id`: Shaper profile ID that needs to be deleted.

4.10.8 Add port traffic management shared shaper

Create the port traffic management shared shaper:

```
testpmd> add port tm node shared shaper (port_id) (shared_shaper_id) \
(shaper_profile_id)
```

where:

- `shared_shaper_id`: Shared shaper ID to be created.
- `shaper_profile id`: Shaper profile ID for shared shaper.

4.10.9 Set port traffic management shared shaper

Update the port traffic management shared shaper:

```
testpmd> set port tm node shared shaper (port_id) (shared_shaper_id) \
(shaper_profile_id)
```

where:

- `shared_shaper_id`: Shared shaper ID to be update.
- `shaper_profile id`: Shaper profile ID for shared shaper.

4.10.10 Delete port traffic management shared shaper

Delete the port traffic management shared shaper:

```
testpmd> del port tm node shared shaper (port_id) (shared_shaper_id)
```

where:

- `shared_shaper_id`: Shared shaper ID to be deleted.

4.10.11 Set port traffic management hiearchy node private shaper

set the port traffic management hierarchy node private shaper:

```
testpmd> set port tm node shaper profile (port_id) (node_id) \
(shaper_profile_id)
```

where:

- `shaper_profile id`: Private shaper profile ID to be enabled on the hierarchy node.

4.10.12 Add port traffic management WRED profile

Create a new WRED profile:

```
testpmd> add port tm node wred profile (port_id) (wred_profile_id) \
(color_g) (min_th_g) (max_th_g) (maxp_inv_g) (wq_log2_g) \
(color_y) (min_th_y) (max_th_y) (maxp_inv_y) (wq_log2_y) \
(color_r) (min_th_r) (max_th_r) (maxp_inv_r) (wq_log2_r)
```

where:

- `wred_profile id`: Identifier for the newly create WRED profile
- `color_g`: Packet color (green)
- `min_th_g`: Minimum queue threshold for packet with green color
- `max_th_g`: Minimum queue threshold for packet with green color
- `maxp_inv_g`: Inverse of packet marking probability maximum value (maxp)
- `wq_log2_g`: Negated log2 of queue weight (wq)
- `color_y`: Packet color (yellow)
- `min_th_y`: Minimum queue threshold for packet with yellow color
- `max_th_y`: Minimum queue threshold for packet with yellow color
- `maxp_inv_y`: Inverse of packet marking probability maximum value (maxp)
- `wq_log2_y`: Negated log2 of queue weight (wq)
- `color_r`: Packet color (red)
- `min_th_r`: Minimum queue threshold for packet with yellow color
- `max_th_r`: Minimum queue threshold for packet with yellow color
- `maxp_inv_r`: Inverse of packet marking probability maximum value (maxp)
- `wq_log2_r`: Negated log2 of queue weight (wq)

4.10.13 Delete port traffic management WRED profile

Delete the WRED profile:

```
testpmd> del port tm node wred profile (port_id) (wred_profile_id)
```

4.10.14 Add port traffic management hierarchy nonleaf node

Add nonleaf node to port traffic management hiearchy:

```
testpmd> add port tm nonleaf node (port_id) (node_id) (parent_node_id) \
(priority) (weight) (level_id) (shaper_profile_id) \
(n_sp_priorities) (stats_mask) (n_shared_shapers) \
[(shared_shaper_0) (shared_shaper_1) ...] \
```

where:

- parent_node_id: Node ID of the parent.
- priority: Node priority (highest node priority is zero). This is used by the SP algorithm running on the parent node for scheduling this node.
- weight: Node weight (lowest weight is one). The node weight is relative to the weight sum of all siblings that have the same priority. It is used by the WFQ algorithm running on the parent node for scheduling this node.
- level_id: Hiearchy level of the node.
- shaper_profile_id: Shaper profile ID of the private shaper to be used by the node.
- n_sp_priorities: Number of strict priorities.
- stats_mask: Mask of statistics counter types to be enabled for this node.
- n_shared_shapers: Number of shared shapers.
- shared_shaper_id: Shared shaper id.

4.10.15 Add port traffic management hierarchy leaf node

Add leaf node to port traffic management hiearchy:

```
testpmd> add port tm leaf node (port_id) (node_id) (parent_node_id) \
(priority) (weight) (level_id) (shaper_profile_id) \
(cman_mode) (wred_profile_id) (stats_mask) (n_shared_shapers) \
[(shared_shaper_id) (shared_shaper_id) ...] \
```

where:

- parent_node_id: Node ID of the parent.
- priority: Node priority (highest node priority is zero). This is used by the SP algorithm running on the parent node for scheduling this node.
- weight: Node weight (lowest weight is one). The node weight is relative to the weight sum of all siblings that have the same priority. It is used by the WFQ algorithm running on the parent node for scheduling this node.
- level_id: Hiearchy level of the node.
- shaper_profile_id: Shaper profile ID of the private shaper to be used by the node.
- cman_mode: Congestion management mode to be enabled for this node.
- wred_profile_id: WRED profile id to be enabled for this node.
- stats_mask: Mask of statistics counter types to be enabled for this node.
- n_shared_shapers: Number of shared shapers.
- shared_shaper_id: Shared shaper id.

4.10.16 Delete port traffic management hierarchy node

Delete node from port traffic management hiearchy:

```
testpmd> del port tm node (port_id) (node_id)
```

4.10.17 Update port traffic management hierarchy parent node

Update port traffic management hierarchy parent node:

```
testpmd> set port tm node parent (port_id) (node_id) (parent_node_id) \
(priority) (weight)
```

This function can only be called after the hierarchy commit invocation. Its success depends on the port support for this operation, as advertised through the port capability set. This function is valid for all nodes of the traffic management hierarchy except root node.

4.10.18 Suspend port traffic management hierarchy node

testpmd> suspend port tm node (port_id) (node_id)

4.10.19 Resume port traffic management hierarchy node

testpmd> resume port tm node (port_id) (node_id)

4.10.20 Commit port traffic management hierarchy

Commit the traffic management hierarchy on the port:

```
testpmd> port tm hierarchy commit (port_id) (clean_on_fail)
```

where:

- `clean_on_fail`: When set to non-zero, hierarchy is cleared on function call failure. On the other hand, hierarchy is preserved when this parameter is equal to zero.

4.10.21 Set port traffic management default hierarchy (softnic forwarding mode)

set the traffic management default hierarchy on the port:

```
testpmd> set port tm hierarchy default (port_id)
```

4.11 Filter Functions

This section details the available filter functions that are available.

Note these functions interface the deprecated legacy filtering framework, superseded by *rte_flow*. See *Flow rules management*.

4.11.1 ethertype_filter

Add or delete a L2 Ethertype filter, which identify packets by their L2 Ethertype mainly assign them to a receive queue:

```
ethertype_filter (port_id) (add|del) (mac_addr|mac_ignr) (mac_address) \
                 ethertype (ether_type) (drop|fwd) queue (queue_id)
```

The available information parameters are:

- port_id: The port which the Ethertype filter assigned on.
- mac_addr: Compare destination mac address.
- mac_ignr: Ignore destination mac address match.
- mac_address: Destination mac address to match.
- ether_type: The EtherType value want to match, for example 0x0806 for ARP packet. 0x0800 (IPv4) and 0x86DD (IPv6) are invalid.
- queue_id: The receive queue associated with this EtherType filter. It is meaningless when deleting or dropping.

Example, to add/remove an ethertype filter rule:

```
testpmd> ethertype_filter 0 add mac_ignr 00:11:22:33:44:55 \
                          ethertype 0x0806 fwd queue 3

testpmd> ethertype_filter 0 del mac_ignr 00:11:22:33:44:55 \
                          ethertype 0x0806 fwd queue 3
```

4.11.2 2tuple_filter

Add or delete a 2-tuple filter, which identifies packets by specific protocol and destination TCP/UDP port and forwards packets into one of the receive queues:

```
2tuple_filter (port_id) (add|del) dst_port (dst_port_value) \
              protocol (protocol_value) mask (mask_value) \
              tcp_flags (tcp_flags_value) priority (prio_value) \
              queue (queue_id)
```

The available information parameters are:

- port_id: The port which the 2-tuple filter assigned on.
- dst_port_value: Destination port in L4.
- protocol_value: IP L4 protocol.
- mask_value: Participates in the match or not by bit for field above, 1b means participate.
- tcp_flags_value: TCP control bits. The non-zero value is invalid, when the pro_value is not set to 0x06 (TCP).
- prio_value: Priority of this filter.
- queue_id: The receive queue associated with this 2-tuple filter.

Example, to add/remove an 2tuple filter rule:

```
testpmd> 2tuple_filter 0 add dst_port 32 protocol 0x06 mask 0x03 \
                       tcp_flags 0x02 priority 3 queue 3

testpmd> 2tuple_filter 0 del dst_port 32 protocol 0x06 mask 0x03 \
                       tcp_flags 0x02 priority 3 queue 3
```

4.11.3 5tuple_filter

Add or delete a 5-tuple filter, which consists of a 5-tuple (protocol, source and destination IP addresses, source and destination TCP/UDP/SCTP port) and routes packets into one of the receive queues:

```
5tuple_filter (port_id) (add|del) dst_ip (dst_address) src_ip \
              (src_address) dst_port (dst_port_value) \
              src_port (src_port_value) protocol (protocol_value) \
              mask (mask_value) tcp_flags (tcp_flags_value) \
              priority (prio_value) queue (queue_id)
```

The available information parameters are:

- `port_id`: The port which the 5-tuple filter assigned on.
- `dst_address`: Destination IP address.
- `src_address`: Source IP address.
- `dst_port_value`: TCP/UDP destination port.
- `src_port_value`: TCP/UDP source port.
- `protocol_value`: L4 protocol.
- `mask_value`: Participates in the match or not by bit for field above, 1b means participate
- `tcp_flags_value`: TCP control bits. The non-zero value is invalid, when the protocol_value is not set to 0x06 (TCP).
- `prio_value`: The priority of this filter.
- `queue_id`: The receive queue associated with this 5-tuple filter.

Example, to add/remove an 5tuple filter rule:

```
testpmd> 5tuple_filter 0 add dst_ip 2.2.2.5 src_ip 2.2.2.4 \
         dst_port 64 src_port 32 protocol 0x06 mask 0x1F \
         flags 0x0 priority 3 queue 3

testpmd> 5tuple_filter 0 del dst_ip 2.2.2.5 src_ip 2.2.2.4 \
         dst_port 64 src_port 32 protocol 0x06 mask 0x1F \
         flags 0x0 priority 3 queue 3
```

4.11.4 syn_filter

Using the SYN filter, TCP packets whose *SYN* flag is set can be forwarded to a separate queue:

```
syn_filter (port_id) (add|del) priority (high|low) queue (queue_id)
```

The available information parameters are:

- `port_id`: The port which the SYN filter assigned on.
- `high`: This SYN filter has higher priority than other filters.
- `low`: This SYN filter has lower priority than other filters.
- `queue_id`: The receive queue associated with this SYN filter

Example:

```
testpmd> syn_filter 0 add priority high queue 3
```

4.11.5 flex_filter

With flex filter, packets can be recognized by any arbitrary pattern within the first 128 bytes of the packet and routed into one of the receive queues:

```
flex_filter (port_id) (add|del) len (len_value) bytes (bytes_value) \
            mask (mask_value) priority (prio_value) queue (queue_id)
```

The available information parameters are:

- `port_id`: The port which the Flex filter is assigned on.
- `len_value`: Filter length in bytes, no greater than 128.
- `bytes_value`: A string in hexadecimal, means the value the flex filter needs to match.
- `mask_value`: A string in hexadecimal, bit 1 means corresponding byte participates in the match.
- `prio_value`: The priority of this filter.
- `queue_id`: The receive queue associated with this Flex filter.

Example:

```
testpmd> flex_filter 0 add len 16 bytes 0x00000000000000000000000008060000 \
                       mask 000C priority 3 queue 3

testpmd> flex_filter 0 del len 16 bytes 0x00000000000000000000000008060000 \
                       mask 000C priority 3 queue 3
```

4.11.6 flow_director_filter

The Flow Director works in receive mode to identify specific flows or sets of flows and route them to specific queues.

Four types of filtering are supported which are referred to as Perfect Match, Signature, Perfect-mac-vlan and Perfect-tunnel filters, the match mode is set by the `--pkt-filter-mode` command-line parameter:

- Perfect match filters. The hardware checks a match between the masked fields of the received packets and the programmed filters. The masked fields are for IP flow.
- Signature filters. The hardware checks a match between a hash-based signature of the masked fields of the received packet.
- Perfect-mac-vlan match filters. The hardware checks a match between the masked fields of the received packets and the programmed filters. The masked fields are for MAC VLAN flow.
- Perfect-tunnel match filters. The hardware checks a match between the masked fields of the received packets and the programmed filters. The masked fields are for tunnel flow.
- Perfect-raw-flow-type match filters. The hardware checks a match between the masked fields of the received packets and pre-loaded raw (template) packet. The masked fields are specified by input sets.

The Flow Director filters can match the different fields for different type of packet: flow type, specific input set per flow type and the flexible payload.

The Flow Director can also mask out parts of all of these fields so that filters are only applied to certain fields or parts of the fields.

Note that for raw flow type mode the source and destination fields in the raw packet buffer need to be presented in a reversed order with respect to the expected received packets. For example: IP source and destination addresses or TCP/UDP/SCTP source and destination ports

Different NICs may have different capabilities, command show port fdir (port_id) can be used to acquire the information.

Commands to add flow director filters of different flow types:

```
flow_director_filter (port_id) mode IP (add|del|update) \
                     flow (ipv4-other|ipv4-frag|ipv6-other|ipv6-frag) \
                     src (src_ip_address) dst (dst_ip_address) \
                     tos (tos_value) proto (proto_value) ttl (ttl_value) \
                     vlan (vlan_value) flexbytes (flexbytes_value) \
                     (drop|fwd) pf|vf(vf_id) queue (queue_id) \
                     fd_id (fd_id_value)

flow_director_filter (port_id) mode IP (add|del|update) \
                     flow (ipv4-tcp|ipv4-udp|ipv6-tcp|ipv6-udp) \
                     src (src_ip_address) (src_port) \
                     dst (dst_ip_address) (dst_port) \
                     tos (tos_value) ttl (ttl_value) \
                     vlan (vlan_value) flexbytes (flexbytes_value) \
                     (drop|fwd) queue pf|vf(vf_id) (queue_id) \
                     fd_id (fd_id_value)

flow_director_filter (port_id) mode IP (add|del|update) \
                     flow (ipv4-sctp|ipv6-sctp) \
                     src (src_ip_address) (src_port) \
                     dst (dst_ip_address) (dst_port) \
                     tos (tos_value) ttl (ttl_value) \
                     tag (verification_tag) vlan (vlan_value) \
                     flexbytes (flexbytes_value) (drop|fwd) \
                     pf|vf(vf_id) queue (queue_id) fd_id (fd_id_value)

flow_director_filter (port_id) mode IP (add|del|update) flow l2_payload \
                     ether (ethertype) flexbytes (flexbytes_value) \
                     (drop|fwd) pf|vf(vf_id) queue (queue_id)
                     fd_id (fd_id_value)

flow_director_filter (port_id) mode MAC-VLAN (add|del|update) \
                     mac (mac_address) vlan (vlan_value) \
                     flexbytes (flexbytes_value) (drop|fwd) \
                     queue (queue_id) fd_id (fd_id_value)

flow_director_filter (port_id) mode Tunnel (add|del|update) \
                     mac (mac_address) vlan (vlan_value) \
                     tunnel (NVGRE|VxLAN) tunnel-id (tunnel_id_value) \
                     flexbytes (flexbytes_value) (drop|fwd) \
                     queue (queue_id) fd_id (fd_id_value)

flow_director_filter (port_id) mode raw (add|del|update) flow (flow_id) \
                     (drop|fwd) queue (queue_id) fd_id (fd_id_value) \
                     packet (packet file name)
```

For example, to add an ipv4-udp flow type filter:

```
testpmd> flow_director_filter 0 mode IP add flow ipv4-udp src 2.2.2.3 32 \
         dst 2.2.2.5 33 tos 2 ttl 40 vlan 0x1 flexbytes (0x88,0x48) \
```

```
          fwd pf queue 1 fd_id 1
```

For example, add an ipv4-other flow type filter:

```
testpmd> flow_director_filter 0 mode IP add flow ipv4-other src 2.2.2.3 \
          dst 2.2.2.5 tos 2 proto 20 ttl 40 vlan 0x1 \
          flexbytes (0x88,0x48) fwd pf queue 1 fd_id 1
```

4.11.7 flush_flow_director

Flush all flow director filters on a device:

```
testpmd> flush_flow_director (port_id)
```

Example, to flush all flow director filter on port 0:

```
testpmd> flush_flow_director 0
```

4.11.8 flow_director_mask

Set flow director's input masks:

```
flow_director_mask (port_id) mode IP vlan (vlan_value) \
                   src_mask (ipv4_src) (ipv6_src) (src_port) \
                   dst_mask (ipv4_dst) (ipv6_dst) (dst_port)

flow_director_mask (port_id) mode MAC-VLAN vlan (vlan_value)

flow_director_mask (port_id) mode Tunnel vlan (vlan_value) \
                   mac (mac_value) tunnel-type (tunnel_type_value) \
                   tunnel-id (tunnel_id_value)
```

Example, to set flow director mask on port 0:

```
testpmd> flow_director_mask 0 mode IP vlan 0xefff \
         src_mask 255.255.255.255 \
             FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF 0xFFFF \
         dst_mask 255.255.255.255 \
             FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF:FFFF 0xFFFF
```

4.11.9 flow_director_flex_mask

set masks of flow director's flexible payload based on certain flow type:

```
testpmd> flow_director_flex_mask (port_id) \
         flow (none|ipv4-other|ipv4-frag|ipv4-tcp|ipv4-udp|ipv4-sctp| \
               ipv6-other|ipv6-frag|ipv6-tcp|ipv6-udp|ipv6-sctp| \
               l2_payload|all) (mask)
```

Example, to set flow director's flex mask for all flow type on port 0:

```
testpmd> flow_director_flex_mask 0 flow all \
         (0xff,0xff,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
```

4.11.10 flow_director_flex_payload

Configure flexible payload selection:

```
flow_director_flex_payload (port_id) (raw|l2|l3|l4) (config)
```

For example, to select the first 16 bytes from the offset 4 (bytes) of packet's payload as flexible payload:

```
testpmd> flow_director_flex_payload 0 l4 \
         (4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19)
```

4.11.11 get_sym_hash_ena_per_port

Get symmetric hash enable configuration per port:

```
get_sym_hash_ena_per_port (port_id)
```

For example, to get symmetric hash enable configuration of port 1:

```
testpmd> get_sym_hash_ena_per_port 1
```

4.11.12 set_sym_hash_ena_per_port

Set symmetric hash enable configuration per port to enable or disable:

```
set_sym_hash_ena_per_port (port_id) (enable|disable)
```

For example, to set symmetric hash enable configuration of port 1 to enable:

```
testpmd> set_sym_hash_ena_per_port 1 enable
```

4.11.13 get_hash_global_config

Get the global configurations of hash filters:

```
get_hash_global_config (port_id)
```

For example, to get the global configurations of hash filters of port 1:

```
testpmd> get_hash_global_config 1
```

4.11.14 set_hash_global_config

Set the global configurations of hash filters:

```
set_hash_global_config (port_id) (toeplitz|simple_xor|default) \
(ipv4|ipv4-frag|ipv4-tcp|ipv4-udp|ipv4-sctp|ipv4-other|ipv6|ipv6-frag| \
ipv6-tcp|ipv6-udp|ipv6-sctp|ipv6-other|l2_payload|<flow_id>) \
(enable|disable)
```

For example, to enable simple_xor for flow type of ipv6 on port 2:

```
testpmd> set_hash_global_config 2 simple_xor ipv6 enable
```

4.11.15 set_hash_input_set

Set the input set for hash:

```
set_hash_input_set (port_id) (ipv4-frag|ipv4-tcp|ipv4-udp|ipv4-sctp| \
ipv4-other|ipv6-frag|ipv6-tcp|ipv6-udp|ipv6-sctp|ipv6-other| \
l2_payload|<flow_id>) (ovlan|ivlan|src-ipv4|dst-ipv4|src-ipv6|dst-ipv6| \
ipv4-tos|ipv4-proto|ipv6-tc|ipv6-next-header|udp-src-port|udp-dst-port| \
tcp-src-port|tcp-dst-port|sctp-src-port|sctp-dst-port|sctp-veri-tag| \
```

```
udp-key|gre-key|fld-1st|fld-2nd|fld-3rd|fld-4th|fld-5th|fld-6th|fld-7th| \
fld-8th|none) (select|add)
```

For example, to add source IP to hash input set for flow type of ipv4-udp on port 0:

```
testpmd> set_hash_input_set 0 ipv4-udp src-ipv4 add
```

4.11.16 set_fdir_input_set

The Flow Director filters can match the different fields for different type of packet, i.e. specific input set on per flow type and the flexible payload. This command can be used to change input set for each flow type.

Set the input set for flow director:

```
set_fdir_input_set (port_id) (ipv4-frag|ipv4-tcp|ipv4-udp|ipv4-sctp| \
ipv4-other|ipv6|ipv6-frag|ipv6-tcp|ipv6-udp|ipv6-sctp|ipv6-other| \
l2_payload|<flow_id>) (ivlan|ethertype|src-ipv4|dst-ipv4|src-ipv6|dst-ipv6| \
ipv4-tos|ipv4-proto|ipv4-ttl|ipv6-tc|ipv6-next-header|ipv6-hop-limits| \
tudp-src-port|udp-dst-port|cp-src-port|tcp-dst-port|sctp-src-port| \
sctp-dst-port|sctp-veri-tag|none) (select|add)
```

For example to add source IP to FD input set for flow type of ipv4-udp on port 0:

```
testpmd> set_fdir_input_set 0 ipv4-udp src-ipv4 add
```

4.11.17 global_config

Set different GRE key length for input set:

```
global_config (port_id) gre-key-len (number in bytes)
```

For example to set GRE key length for input set to 4 bytes on port 0:

```
testpmd> global_config 0 gre-key-len 4
```

4.12 Flow rules management

Control of the generic flow API (*rte_flow*) is fully exposed through the `flow` command (validation, creation, destruction, queries and operation modes).

Considering *rte_flow* overlaps with all *Filter Functions*, using both features simultaneously may cause undefined side-effects and is therefore not recommended.

4.12.1 `flow` syntax

Because the `flow` command uses dynamic tokens to handle the large number of possible flow rules combinations, its behavior differs slightly from other commands, in particular:

- Pressing *?* or the *<tab>* key displays contextual help for the current token, not that of the entire command.
- Optional and repeated parameters are supported (provided they are listed in the contextual help).

The first parameter stands for the operation mode. Possible operations and their general syntax are described below. They are covered in detail in the following sections.

- Check whether a flow rule can be created:

```
flow validate {port_id}
    [group {group_id}] [priority {level}] [ingress] [egress] [transfer]
    pattern {item} [/ {item} [...]] / end
    actions {action} [/ {action} [...]] / end
```

- Create a flow rule:

```
flow create {port_id}
    [group {group_id}] [priority {level}] [ingress] [egress] [transfer]
    pattern {item} [/ {item} [...]] / end
    actions {action} [/ {action} [...]] / end
```

- Destroy specific flow rules:

```
flow destroy {port_id} rule {rule_id} [...]
```

- Destroy all flow rules:

```
flow flush {port_id}
```

- Query an existing flow rule:

```
flow query {port_id} {rule_id} {action}
```

- List existing flow rules sorted by priority, filtered by group identifiers:

```
flow list {port_id} [group {group_id}] [...]
```

- Restrict ingress traffic to the defined flow rules:

```
flow isolate {port_id} {boolean}
```

4.12.2 Validating flow rules

`flow validate` reports whether a flow rule would be accepted by the underlying device in its current state but stops short of creating it. It is bound to `rte_flow_validate()`:

```
flow validate {port_id}
   [group {group_id}] [priority {level}] [ingress] [egress] [transfer]
   pattern {item} [/ {item} [...]] / end
   actions {action} [/ {action} [...]] / end
```

If successful, it will show:

```
Flow rule validated
```

Otherwise it will show an error message of the form:

```
Caught error type [...] ([...]): [...]
```

This command uses the same parameters as `flow create`, their format is described in *Creating flow rules*.

Check whether redirecting any Ethernet packet received on port 0 to RX queue index 6 is supported:

```
testpmd> flow validate 0 ingress pattern eth / end
   actions queue index 6 / end
Flow rule validated
testpmd>
```

Port 0 does not support TCPv6 rules:

```
testpmd> flow validate 0 ingress pattern eth / ipv6 / tcp / end
   actions drop / end
Caught error type 9 (specific pattern item): Invalid argument
testpmd>
```

4.12.3 Creating flow rules

`flow create` validates and creates the specified flow rule. It is bound to `rte_flow_create()`:

```
flow create {port_id}
   [group {group_id}] [priority {level}] [ingress] [egress] [transfer]
   pattern {item} [/ {item} [...]] / end
   actions {action} [/ {action} [...]] / end
```

If successful, it will return a flow rule ID usable with other commands:

```
Flow rule #[...] created
```

Otherwise it will show an error message of the form:

```
Caught error type [...] ([...]): [...]
```

Parameters describe in the following order:

- Attributes (*group*, *priority*, *ingress*, *egress*, *transfer* tokens).
- A matching pattern, starting with the *pattern* token and terminated by an *end* pattern item.
- Actions, starting with the *actions* token and terminated by an *end* action.

These translate directly to *rte_flow* objects provided as-is to the underlying functions.

The shortest valid definition only comprises mandatory tokens:

```
testpmd> flow create 0 pattern end actions end
```

Note that PMDs may refuse rules that essentially do nothing such as this one.

All unspecified object values are automatically initialized to 0.

Attributes

These tokens affect flow rule attributes (`struct rte_flow_attr`) and are specified before the `pattern` token.

- `group {group id}`: priority group.
- `priority {level}`: priority level within group.
- `ingress`: rule applies to ingress traffic.
- `egress`: rule applies to egress traffic.
- `transfer`: apply rule directly to endpoints found in pattern.

Each instance of an attribute specified several times overrides the previous value as shown below (group 4 is used):

```
testpmd> flow create 0 group 42 group 24 group 4 [...]
```

Note that once enabled, `ingress` and `egress` cannot be disabled.

While not specifying a direction is an error, some rules may allow both simultaneously.

Most rules affect RX therefore contain the `ingress` token:

```
testpmd> flow create 0 ingress pattern [...]
```

Matching pattern

A matching pattern starts after the `pattern` token. It is made of pattern items and is terminated by a mandatory `end` item.

Items are named after their type (*RTE_FLOW_ITEM_TYPE_* from `enum rte_flow_item_type`).

The `/` token is used as a separator between pattern items as shown below:

```
testpmd> flow create 0 ingress pattern eth / ipv4 / udp / end [...]
```

Note that protocol items like these must be stacked from lowest to highest layer to make sense. For instance, the following rule is either invalid or unlikely to match any packet:

```
testpmd> flow create 0 ingress pattern eth / udp / ipv4 / end [...]
```

More information on these restrictions can be found in the *rte_flow* documentation.

Several items support additional specification structures, for example `ipv4` allows specifying source and destination addresses as follows:

```
testpmd> flow create 0 ingress pattern eth / ipv4 src is 10.1.1.1
   dst is 10.2.0.0 / end [...]
```

This rule matches all IPv4 traffic with the specified properties.

In this example, `src` and `dst` are field names of the underlying `struct rte_flow_item_ipv4` object. All item properties can be specified in a similar fashion.

The `is` token means that the subsequent value must be matched exactly, and assigns `spec` and `mask` fields in `struct rte_flow_item` accordingly. Possible assignment tokens are:

- `is`: match value perfectly (with full bit-mask).
- `spec`: match value according to configured bit-mask.
- `last`: specify upper bound to establish a range.
- `mask`: specify bit-mask with relevant bits set to one.
- `prefix`: generate bit-mask from a prefix length.

These yield identical results:

```
ipv4 src is 10.1.1.1

ipv4 src spec 10.1.1.1 src mask 255.255.255.255

ipv4 src spec 10.1.1.1 src prefix 32

ipv4 src is 10.1.1.1 src last 10.1.1.1 # range with a single value

ipv4 src is 10.1.1.1 src last 0 # 0 disables range
```

Inclusive ranges can be defined with `last`:

```
ipv4 src is 10.1.1.1 src last 10.2.3.4 # 10.1.1.1 to 10.2.3.4
```

Note that `mask` affects both `spec` and `last`:

```
ipv4 src is 10.1.1.1 src last 10.2.3.4 src mask 255.255.0.0
   # matches 10.1.0.0 to 10.2.255.255
```

Properties can be modified multiple times:

```
ipv4 src is 10.1.1.1 src is 10.1.2.3 src is 10.2.3.4 # matches 10.2.3.4

ipv4 src is 10.1.1.1 src prefix 24 src prefix 16 # matches 10.1.0.0/16
```

Pattern items

This section lists supported pattern items and their attributes, if any.

- `end`: end list of pattern items.
- `void`: no-op pattern item.
- `invert`: perform actions when pattern does not match.
- `any`: match any protocol for the current layer.
  - `num {unsigned}`: number of layers covered.
- `pf`: match traffic from/to the physical function.
- `vf`: match traffic from/to a virtual function ID.
  - `id {unsigned}`: VF ID.
- `phy_port`: match traffic from/to a specific physical port.
  - `index {unsigned}`: physical port index.
- `port_id`: match traffic from/to a given DPDK port ID.
  - `id {unsigned}`: DPDK port ID.
- `mark`: match value set in previously matched flow rule using the mark action.
  - `id {unsigned}`: arbitrary integer value.
- `raw`: match an arbitrary byte string.
  - `relative {boolean}`: look for pattern after the previous item.
  - `search {boolean}`: search pattern from offset (see also limit).
  - `offset {integer}`: absolute or relative offset for pattern.
  - `limit {unsigned}`: search area limit for start of pattern.
  - `pattern {string}`: byte string to look for.
- `eth`: match Ethernet header.
  - `dst {MAC-48}`: destination MAC.
  - `src {MAC-48}`: source MAC.
  - `type {unsigned}`: EtherType or TPID.
- `vlan`: match 802.1Q/ad VLAN tag.
  - `tci {unsigned}`: tag control information.

  - `pcp {unsigned}`: priority code point.
  - `dei {unsigned}`: drop eligible indicator.
  - `vid {unsigned}`: VLAN identifier.
  - `inner_type {unsigned}`: inner EtherType or TPID.
- `ipv4`: match IPv4 header.
  - `tos {unsigned}`: type of service.
  - `ttl {unsigned}`: time to live.
  - `proto {unsigned}`: next protocol ID.
  - `src {ipv4 address}`: source address.
  - `dst {ipv4 address}`: destination address.
- `ipv6`: match IPv6 header.
  - `tc {unsigned}`: traffic class.
  - `flow {unsigned}`: flow label.
  - `proto {unsigned}`: protocol (next header).
  - `hop {unsigned}`: hop limit.
  - `src {ipv6 address}`: source address.
  - `dst {ipv6 address}`: destination address.
- `icmp`: match ICMP header.
  - `type {unsigned}`: ICMP packet type.
  - `code {unsigned}`: ICMP packet code.
- `udp`: match UDP header.
  - `src {unsigned}`: UDP source port.
  - `dst {unsigned}`: UDP destination port.
- `tcp`: match TCP header.
  - `src {unsigned}`: TCP source port.
  - `dst {unsigned}`: TCP destination port.
- `sctp`: match SCTP header.
  - `src {unsigned}`: SCTP source port.
  - `dst {unsigned}`: SCTP destination port.
  - `tag {unsigned}`: validation tag.
  - `cksum {unsigned}`: checksum.
- `vxlan`: match VXLAN header.
  - `vni {unsigned}`: VXLAN identifier.
- `e_tag`: match IEEE 802.1BR E-Tag header.

  - grp_ecid_b {unsigned}: GRP and E-CID base.
- nvgre: match NVGRE header.
  - tni {unsigned}: virtual subnet ID.
- mpls: match MPLS header.
  - label {unsigned}: MPLS label.
- gre: match GRE header.
  - protocol {unsigned}: protocol type.
- fuzzy: fuzzy pattern match, expect faster than default.
  - thresh {unsigned}: accuracy threshold.
- gtp, gtpc, gtpu: match GTPv1 header.
  - teid {unsigned}: tunnel endpoint identifier.
- geneve: match GENEVE header.
  - vni {unsigned}: virtual network identifier.
  - protocol {unsigned}: protocol type.
- vxlan-gpe: match VXLAN-GPE header.
  - vni {unsigned}: VXLAN-GPE identifier.
- arp_eth_ipv4: match ARP header for Ethernet/IPv4.
  - sha {MAC-48}: sender hardware address.
  - spa {ipv4 address}: sender IPv4 address.
  - tha {MAC-48}: target hardware address.
  - tpa {ipv4 address}: target IPv4 address.
- ipv6_ext: match presence of any IPv6 extension header.
  - next_hdr {unsigned}: next header.
- icmp6: match any ICMPv6 header.
  - type {unsigned}: ICMPv6 type.
  - code {unsigned}: ICMPv6 code.
- icmp6_nd_ns: match ICMPv6 neighbor discovery solicitation.
  - target_addr {ipv6 address}: target address.
- icmp6_nd_na: match ICMPv6 neighbor discovery advertisement.
  - target_addr {ipv6 address}: target address.
- icmp6_nd_opt: match presence of any ICMPv6 neighbor discovery option.
  - type {unsigned}: ND option type.
- icmp6_nd_opt_sla_eth: match ICMPv6 neighbor discovery source Ethernet link-layer address option.
  - sla {MAC-48}: source Ethernet LLA.

- `icmp6_nd_opt_sla_eth`: match ICMPv6 neighbor discovery target Ethernet link-layer address option.
  - `tla {MAC-48}`: target Ethernet LLA.

Actions list

A list of actions starts after the `actions` token in the same fashion as *Matching pattern*; actions are separated by `/` tokens and the list is terminated by a mandatory `end` action.

Actions are named after their type (*RTE_FLOW_ACTION_TYPE_* from `enum rte_flow_action_type`).

Dropping all incoming UDPv4 packets can be expressed as follows:

```
testpmd> flow create 0 ingress pattern eth / ipv4 / udp / end
   actions drop / end
```

Several actions have configurable properties which must be specified when there is no valid default value. For example, `queue` requires a target queue index.

This rule redirects incoming UDPv4 traffic to queue index 6:

```
testpmd> flow create 0 ingress pattern eth / ipv4 / udp / end
   actions queue index 6 / end
```

While this one could be rejected by PMDs (unspecified queue index):

```
testpmd> flow create 0 ingress pattern eth / ipv4 / udp / end
   actions queue / end
```

As defined by *rte_flow*, the list is not ordered, all actions of a given rule are performed simultaneously. These are equivalent:

```
queue index 6 / void / mark id 42 / end
```

```
void / mark id 42 / queue index 6 / end
```

All actions in a list should have different types, otherwise only the last action of a given type is taken into account:

```
queue index 4 / queue index 5 / queue index 6 / end # will use queue 6
```

```
drop / drop / drop / end # drop is performed only once
```

```
mark id 42 / queue index 3 / mark id 24 / end # mark will be 24
```

Considering they are performed simultaneously, opposite and overlapping actions can sometimes be combined when the end result is unambiguous:

```
drop / queue index 6 / end # drop has no effect
```

```
queue index 6 / rss queues 6 7 8 / end # queue has no effect
```

```
drop / passthru / end # drop has no effect
```

Note that PMDs may still refuse such combinations.

Actions

This section lists supported actions and their attributes, if any.

- `end`: end list of actions.

- `void`: no-op action.
- `passthru`: let subsequent rule process matched packets.
- `jump`: redirect traffic to group on device.
  - `group {unsigned}`: group to redirect to.
- `mark`: attach 32 bit value to packets.
  - `id {unsigned}`: 32 bit value to return with packets.
- `flag`: flag packets.
- `queue`: assign packets to a given queue index.
  - `index {unsigned}`: queue index to use.
- `drop`: drop packets (note: passthru has priority).
- `count`: enable counters for this rule.
- `rss`: spread packets among several queues.
  - `func {hash function}`: RSS hash function to apply, allowed tokens are the same as *set_hash_global_config*.
  - `level {unsigned}`: encapsulation level for `types`.
  - `types [{RSS hash type} [...]] end`: specific RSS hash types, allowed tokens are the same as *set_hash_input_set*, except that an empty list does not disable RSS but instead requests unspecified "best-effort" settings.
  - `key {string}`: RSS hash key, overrides `key_len`.
  - `key_len {unsigned}`: RSS hash key length in bytes, can be used in conjunction with `key` to pad or truncate it.
  - `queues [{unsigned} [...]] end`: queue indices to use.
- `pf`: direct traffic to physical function.
- `vf`: direct traffic to a virtual function ID.
  - `original {boolean}`: use original VF ID if possible.
  - `id {unsigned}`: VF ID.
- `phy_port`: direct packets to physical port index.
  - `original {boolean}`: use original port index if possible.
  - `index {unsigned}`: physical port index.
- `port_id`: direct matching traffic to a given DPDK port ID.
  - `original {boolean}`: use original DPDK port ID if possible.
  - `id {unsigned}`: DPDK port ID.
- `of_set_mpls_ttl`: OpenFlow's `OFPAT_SET_MPLS_TTL`.
  - `mpls_ttl`: MPLS TTL.
- `of_dec_mpls_ttl`: OpenFlow's `OFPAT_DEC_MPLS_TTL`.
- `of_set_nw_ttl`: OpenFlow's `OFPAT_SET_NW_TTL`.

    - `nw_ttl`: IP TTL.
- `of_dec_nw_ttl`: OpenFlow's `OFPAT_DEC_NW_TTL`.
- `of_copy_ttl_out`: OpenFlow's `OFPAT_COPY_TTL_OUT`.
- `of_copy_ttl_in`: OpenFlow's `OFPAT_COPY_TTL_IN`.
- `of_pop_vlan`: OpenFlow's `OFPAT_POP_VLAN`.
- `of_push_vlan`: OpenFlow's `OFPAT_PUSH_VLAN`.
    - `ethertype`: Ethertype.
- `of_set_vlan_vid`: OpenFlow's `OFPAT_SET_VLAN_VID`.
    - `vlan_vid`: VLAN id.
- `of_set_vlan_pcp`: OpenFlow's `OFPAT_SET_VLAN_PCP`.
    - `vlan_pcp`: VLAN priority.
- `of_pop_mpls`: OpenFlow's `OFPAT_POP_MPLS`.
    - `ethertype`: Ethertype.
- `of_push_mpls`: OpenFlow's `OFPAT_PUSH_MPLS`.
    - `ethertype`: Ethertype.
- `vxlan_encap`: Performs a VXLAN encapsulation, outer layer configuration is done through *Config VXLAN Encap outer layers*.
- `vxlan_decap`: Performs a decapsulation action by stripping all headers of the VXLAN tunnel network overlay from the matched flow.
- `nvgre_encap`: Performs a NVGRE encapsulation, outer layer configuration is done through *Config NVGRE Encap outer layers*.
- `nvgre_decap`: Performs a decapsulation action by stripping all headers of the NVGRE tunnel network overlay from the matched flow.

4.12.4 Destroying flow rules

`flow destroy` destroys one or more rules from their rule ID (as returned by `flow create`), this command calls `rte_flow_destroy()` as many times as necessary:

```
flow destroy {port_id} rule {rule_id} [...]
```

If successful, it will show:

```
Flow rule #[...] destroyed
```

It does not report anything for rule IDs that do not exist. The usual error message is shown when a rule cannot be destroyed:

```
Caught error type [...] ([...]): [...]
```

`flow flush` destroys all rules on a device and does not take extra arguments. It is bound to `rte_flow_flush()`:

```
flow flush {port_id}
```

Any errors are reported as above.

Creating several rules and destroying them:

```
testpmd> flow create 0 ingress pattern eth / ipv6 / end
   actions queue index 2 / end
Flow rule #0 created
testpmd> flow create 0 ingress pattern eth / ipv4 / end
   actions queue index 3 / end
Flow rule #1 created
testpmd> flow destroy 0 rule 0 rule 1
Flow rule #1 destroyed
Flow rule #0 destroyed
testpmd>
```

The same result can be achieved using `flow flush`:

```
testpmd> flow create 0 ingress pattern eth / ipv6 / end
   actions queue index 2 / end
Flow rule #0 created
testpmd> flow create 0 ingress pattern eth / ipv4 / end
   actions queue index 3 / end
Flow rule #1 created
testpmd> flow flush 0
testpmd>
```

Non-existent rule IDs are ignored:

```
testpmd> flow create 0 ingress pattern eth / ipv6 / end
   actions queue index 2 / end
Flow rule #0 created
testpmd> flow create 0 ingress pattern eth / ipv4 / end
   actions queue index 3 / end
Flow rule #1 created
testpmd> flow destroy 0 rule 42 rule 10 rule 2
testpmd>
testpmd> flow destroy 0 rule 0
Flow rule #0 destroyed
testpmd>
```

4.12.5 Querying flow rules

`flow query` queries a specific action of a flow rule having that ability. Such actions collect information that can be reported using this command. It is bound to `rte_flow_query()`:

```
flow query {port_id} {rule_id} {action}
```

If successful, it will display either the retrieved data for known actions or the following message:

```
Cannot display result for action type [...] ([...])
```

Otherwise, it will complain either that the rule does not exist or that some error occurred:

```
Flow rule #[...] not found
```

```
Caught error type [...] ([...]): [...]
```

Currently only the `count` action is supported. This action reports the number of packets that hit the flow rule and the total number of bytes. Its output has the following format:

```
count:
 hits_set: [...] # whether "hits" contains a valid value
 bytes_set: [...] # whether "bytes" contains a valid value
 hits: [...] # number of packets
 bytes: [...] # number of bytes
```

Querying counters for TCPv6 packets redirected to queue 6:

```
testpmd> flow create 0 ingress pattern eth / ipv6 / tcp / end
   actions queue index 6 / count / end
Flow rule #4 created
testpmd> flow query 0 4 count
count:
 hits_set: 1
 bytes_set: 0
 hits: 386446
 bytes: 0
testpmd>
```

4.12.6 Listing flow rules

`flow list` lists existing flow rules sorted by priority and optionally filtered by group identifiers:

```
flow list {port_id} [group {group_id}] [...]
```

This command only fails with the following message if the device does not exist:

```
Invalid port [...]
```

Output consists of a header line followed by a short description of each flow rule, one per line. There is no output at all when no flow rules are configured on the device:

```
ID      Group   Prio    Attr    Rule
[...]   [...]   [...]   [...]   [...]
```

`Attr` column flags:

- `i` for `ingress`.
- `e` for `egress`.

Creating several flow rules and listing them:

```
testpmd> flow create 0 ingress pattern eth / ipv4 / end
   actions queue index 6 / end
Flow rule #0 created
testpmd> flow create 0 ingress pattern eth / ipv6 / end
   actions queue index 2 / end
Flow rule #1 created
testpmd> flow create 0 priority 5 ingress pattern eth / ipv4 / udp / end
   actions rss queues 6 7 8 end / end
Flow rule #2 created
testpmd> flow list 0
ID      Group   Prio    Attr    Rule
0       0       0       i-      ETH IPV4 => QUEUE
1       0       0       i-      ETH IPV6 => QUEUE
2       0       5       i-      ETH IPV4 UDP => RSS
testpmd>
```

Rules are sorted by priority (i.e. group ID first, then priority level):

```
testpmd> flow list 1
ID      Group   Prio    Attr    Rule
0       0       0       i-      ETH => COUNT
6       0       500     i-      ETH IPV6 TCP => DROP COUNT
5       0       1000    i-      ETH IPV6 ICMP => QUEUE
1       24      0       i-      ETH IPV4 UDP => QUEUE
4       24      10      i-      ETH IPV4 TCP => DROP
3       24      20      i-      ETH IPV4 => DROP
2       24      42      i-      ETH IPV4 UDP => QUEUE
```

```
7       63      0       i-      ETH IPV6 UDP VXLAN => MARK QUEUE
testpmd>
```

Output can be limited to specific groups:

```
testpmd> flow list 1 group 0 group 63
ID      Group   Prio    Attr    Rule
0       0       0       i-      ETH => COUNT
6       0       500     i-      ETH IPV6 TCP => DROP COUNT
5       0       1000    i-      ETH IPV6 ICMP => QUEUE
7       63      0       i-      ETH IPV6 UDP VXLAN => MARK QUEUE
testpmd>
```

4.12.7 Toggling isolated mode

`flow isolate` can be used to tell the underlying PMD that ingress traffic must only be injected from the defined flow rules; that no default traffic is expected outside those rules and the driver is free to assign more resources to handle them. It is bound to `rte_flow_isolate()`:

```
flow isolate {port_id} {boolean}
```

If successful, enabling or disabling isolated mode shows either:

```
Ingress traffic on port [...]
   is now restricted to the defined flow rules
```

Or:

```
Ingress traffic on port [...]
   is not restricted anymore to the defined flow rules
```

Otherwise, in case of error:

```
Caught error type [...] ([...]): [...]
```

Mainly due to its side effects, PMDs supporting this mode may not have the ability to toggle it more than once without reinitializing affected ports first (e.g. by exiting testpmd).

Enabling isolated mode:

```
testpmd> flow isolate 0 true
Ingress traffic on port 0 is now restricted to the defined flow rules
testpmd>
```

Disabling isolated mode:

```
testpmd> flow isolate 0 false
Ingress traffic on port 0 is not restricted anymore to the defined flow rules
testpmd>
```

4.12.8 Sample QinQ flow rules

Before creating QinQ rule(s) the following commands should be issued to enable QinQ:

```
testpmd> port stop 0
testpmd> vlan set qinq on 0
```

The above command sets the inner and outer TPID's to 0x8100.

To change the TPID's the following commands should be used:

```
testpmd> vlan set outer tpid 0xa100 0
testpmd> vlan set inner tpid 0x9100 0
testpmd> port start 0
```

Validate and create a QinQ rule on port 0 to steer traffic to a VF queue in a VM.

```
testpmd> flow validate 0 ingress pattern eth / vlan tci is 123 /
    vlan tci is 456 / end actions vf id 1 / queue index 0 / end
Flow rule #0 validated

testpmd> flow create 0 ingress pattern eth / vlan tci is 4 /
    vlan tci is 456 / end actions vf id 123 / queue index 0 / end
Flow rule #0 created

testpmd> flow list 0
ID      Group   Prio    Attr    Rule
0       0       0       i-      ETH VLAN VLAN=>VF QUEUE
```

Validate and create a QinQ rule on port 0 to steer traffic to a queue on the host.

```
testpmd> flow validate 0 ingress pattern eth / vlan tci is 321 /
     vlan tci is 654 / end actions pf / queue index 0 / end
Flow rule #1 validated

testpmd> flow create 0 ingress pattern eth / vlan tci is 321 /
     vlan tci is 654 / end actions pf / queue index 1 / end
Flow rule #1 created

testpmd> flow list 0
ID      Group   Prio    Attr    Rule
0       0       0       i-      ETH VLAN VLAN=>VF QUEUE
1       0       0       i-      ETH VLAN VLAN=>PF QUEUE
```

4.12.9 Sample VXLAN encapsulation rule

VXLAN encapsulation outer layer has default value pre-configured in testpmd source code, those can be changed by using the following commands

IPv4 VXLAN outer header:

```
testpmd> set vxlan ip-version ipv4 vni 4 udp-src 4 udp-dst 4 ip-src 127.0.0.1
       ip-dst 128.0.0.1 eth-src 11:11:11:11:11:11 eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions vxlan_encap /
       queue index 0 / end

testpmd> set vxlan-with-vlan ip-version ipv4 vni 4 udp-src 4 udp-dst 4 ip-src
        127.0.0.1 ip-dst 128.0.0.1 vlan-tci 34 eth-src 11:11:11:11:11:11
        eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions vxlan_encap /
        queue index 0 / end
```

IPv6 VXLAN outer header:

```
testpmd> set vxlan ip-version ipv6 vni 4 udp-src 4 udp-dst 4 ip-src ::1
       ip-dst ::2222 eth-src 11:11:11:11:11:11 eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions vxlan_encap /
        queue index 0 / end

testpmd> set vxlan-with-vlan ip-version ipv6 vni 4 udp-src 4 udp-dst 4
        ip-src ::1 ip-dst ::2222 vlan-tci 34 eth-src 11:11:11:11:11:11
        eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions vxlan_encap /
        queue index 0 / end
```

4.12.10 Sample NVGRE encapsulation rule

NVGRE encapsulation outer layer has default value pre-configured in testpmd source code, those can be changed by using the following commands

IPv4 NVGRE outer header:

```
testpmd> set nvgre ip-version ipv4 tni 4 ip-src 127.0.0.1 ip-dst 128.0.0.1
       eth-src 11:11:11:11:11:11 eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions nvgre_encap /
       queue index 0 / end

testpmd> set nvgre-with-vlan ip-version ipv4 tni 4 ip-src 127.0.0.1
        ip-dst 128.0.0.1 vlan-tci 34 eth-src 11:11:11:11:11:11
        eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions nvgre_encap /
        queue index 0 / end
```

IPv6 NVGRE outer header:

```
testpmd> set nvgre ip-version ipv6 tni 4 ip-src ::1 ip-dst ::2222
       eth-src 11:11:11:11:11:11 eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions nvgre_encap /
       queue index 0 / end

testpmd> set nvgre-with-vlan ip-version ipv6 tni 4 ip-src ::1 ip-dst ::2222
       vlan-tci 34 eth-src 11:11:11:11:11:11 eth-dst 22:22:22:22:22:22
testpmd> flow create 0 ingress pattern end actions nvgre_encap /
       queue index 0 / end
```

4.13 BPF Functions

The following sections show functions to load/unload eBPF based filters.

4.13.1 bpf-load

Load an eBPF program as a callback for partciular RX/TX queue:

```
testpmd> bpf-load rx|tx (portid) (queueid) (load-flags) (bpf-prog-filename)
```

The available load-flags are:

- `J`: use JIT generated native code, otherwise BPF interpreter will be used.
- `M`: assume input parameter is a pointer to rte_mbuf, otherwise assume it is a pointer to first segment's data.
- `-`: none.

Note: You'll need clang v3.7 or above to build bpf program you'd like to load

For example:

```
cd test/bpf
clang -O2 -target bpf -c t1.c
```

Then to load (and JIT compile) t1.o at RX queue 0, port 1:

```
.. code-block:: console
```

```
testpmd> bpf-load rx 1 0 J ./dpdk.org/test/bpf/t1.o
```

To load (not JITed) t1.o at TX queue 0, port 0:

```
.. code-block:: console

testpmd> bpf-load tx 0 0 - ./dpdk.org/test/bpf/t1.o
```

4.13.2 bpf-unload

Unload previously loaded eBPF program for partciular RX/TX queue:

```
testpmd> bpf-unload rx|tx (portid) (queueid)
```

For example to unload BPF filter from TX queue 0, port 0:

```
testpmd> bpf-load tx 0 0 - ./dpdk.org/test/bpf/t1.o
```