Pktgen Documentation

Release 3.2.4

Keith Wiles

Aug 28, 2019

Contents

1 Getting Started with Pktgen 3

2 Running Pktgen 9

3 EAL Commandline Options 13

4 Pktgen Commandline Options 15

5 Multiple Instances of Pktgen or other application 19

6 Pktgen command line directory format 23

7 Runtime Options and Commands 25

8 CLI Sample Application 35

9 CLI library guide 47

10 Running Script Files 59

11 Using Lua with Pktgen 61

12 Socket Support for Pktgen 67

13 Changes in Pktgen 69

14 Pktgen-DPDK - Traffic Generator powered by DPDK 71

15 Copyright and License 79

16 Third Party License Notices 81

Pktgen, (*Packet Gen*-erator) is a software based traffic generator powered by the DPDK fast packet processing framework.

Some of the features of Pktgen are:

- It is capable of generating 10Gbit wire rate traffic with 64 byte frames.
- It can act as a transmitter or receiver at line rate.
- It has a runtime environment to configure, and start and stop traffic flows.
- It can display real time metrics for a number of ports.
- It can generate packets in sequence by iterating source or destination MAC, IP addresses or ports.
- It can handle packets with UDP, TCP, ARP, ICMP, GRE, MPLS and Queue-in-Queue.
- It can be controlled remotely over a TCP connection.
- It is configurable via Lua and can run command scripts to set up repeatable test cases.
- The software is fully available under a BSD licence.

Pktgen was created 2010 by Keith Wiles @ windriver.com, now at intel.com

CHAPTER 1

Getting Started with Pktgen

This section contains instructions on how to get up and running with DPDK and the `pktgen` traffic generator application.

These instructions relate to setting up DPDK and `pktgen` on an Ubuntu desktop system. However, the should work on any recent Linux system with kernel support for hugeTLB/hugepages.

1.1 System requirements

The main system requirement is that the DPDK packet processing framework is supported.

The DPDK Linux Getting Started Guide has a section on the System Requirements that explains the BIOS, System and Toolchain requirements to compile and run a DPDK based application such as `pktgen`. Ensure that your system meets those requirements before proceeding.

You will also need a DPDK supported NIC.

The current version of `pktgen` was developed and tested using Ubuntu 13.10 x86_64, kernel version 3.5.0-25, on a Westmere Dual socket board running at 2.4GHz with 12GB of ram 6GB per socket.

1.2 Setting up hugeTLB/hugepage support

To get hugeTLB/hugepage support your Linux kernel must be at least 2.6.33 and the `HUGETLBFS` kernel option must be enabled.

The DPDK Linux Getting Started Guide has a section on the Use of Hugepages in the Linux Environment.

Once you have made the required changed make sure you have HUGE TLB support in the kernel with the following commands:

```
$ grep -i huge /boot/config-2.6.35-24-generic
CONFIG_HUGETLBFS=y
CONFIG_HUGETLB_PAGE=y

$ grep -i huge /proc/meminfo

HugePages_Total:      128
HugePages_Free:       128
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
```

The values in Total and Free may be different depending on your system.

You will need to edit the `/etc/sysctl.conf` file to setup the hugepages size:

```
$ sudo vi /etc/sysctl.conf
Add to the bottom of the file:
vm.nr_hugepages=256
```

You can configure the `vm.nr_hugepages=256` as required. In some cases making it too small will effect the performance of pktgen or cause it to terminate on startup.

You will also need to edit the `/etc/fstab` file to mount the hugepages at startup:

```
$ sudo vi /etc/fstab
Add to the bottom of the file:
huge /mnt/huge hugetlbfs defaults 0 0

$ sudo mkdir /mnt/huge
$ sudo chmod 777 /mnt/huge
```

You should also reboot your machine as the huge pages must be setup just after boot to make sure there is enough contiguous memory for the 2MB pages.

Note: If you start an application that makes extensive use of hugepages, such as Eclipse or WR Workbench, before starting `pktgen` for the first time after reboot, `pktgen` may fail to load. In this case you should close the other application that is using hugepages.

1.3 BIOS settings

In the BIOS make sure that the HPET High Precision Event Timer is enabled. Also make sure hyper-threading is enabled. See the DPDK documentation on enabling additional BIOS functionality for more details.

1.4 Terminal display

The `pktgen` output display requires 132 columns and about 42 lines to display correctly. The author uses an xterm of 132x42, but you can also have a larger display and maybe a bit smaller. If you are displaying more then 4-6 ports then you will need a wider display.

Pktgen allows you to view a set ports via the `page` runtime command if they do not all fit on the screen at one time, see commands.

Pktgen uses VT100 control codes display its output screens, which means your terminal must support VT100.

It is also best to set your terminal background to black when working with the default `pktgen` color scheme.

1.5 Get the source code

Pktgen requires the DPDK source code to build.

The main `dpdk` and `pktgen` git repositories are hosted on dpdk.org.

The `dpdk` code can be cloned as follows:

```
git clone git://dpdk.org/dpdk
# or:
git clone http://dpdk.org/git/dpdk
```

The `pktgen` code can be cloned as follows:

```
git clone git://dpdk.org/apps/pktgen-dpdk
# or:
git clone http://dpdk.org/git/apps/pktgen-dpdk
```

In the instructions below the repository close directories are referred to as `DPDKInstallDir` and `PktgenInstallDir`.

You will also require the Linux kernel headers to allow DPDK to build its kernel modules. On Ubuntu you can install them as follows (where the version matches the kernel version):

```
$ sudo apt-get install linux-headers-3.5.0-32-generic
```

DPDK can also work with a `libpcap` driver which is sometimes useful for testing without a real NIC or for low speed packet capture. Install the `libpcap` development libs using your package manage. For example:

```
$ sudo apt-get install libpcap-dev
```

1.6 Build DPDK and Pktgen

Set up the environmental variables required by DPDK:

```
export RTE_SDK=<DPDKInstallDir>
export RTE_TARGET=x86_64-native-linux-gcc
or
export RTE_TARGET=x86_64-native-linuxapp-gcc

# or use clang if you have it installed:
export RTE_TARGET=x86_64-native-linux-clang
or
export RTE_TARGET=x86_64-native-linuxapp-clang
```

Create the DPDK build tree:

```
$ cd $RTE_SDK
$ make install T=x86_64-native-linux-gcc
or
$ make install T=x86_64-native-linuxapp-gcc
```

This above command will create the *x86_64-pktgen-linux-gcc* directory in the top level of the $RTE_SDK directory. It will also build the basic DPDK libraries, kernel modules and build tree.

Pktgen can then be built as follows:

```
$ cd <PktgenInstallDir>
$ make
```

1.7 Setting up your environment

In the `PktgenInstallDir`/tools level directory there is `run.py` script, which should be run once per boot with the -s option to setup the ports. The same configuration file is also used to run pktgen by removing the -s option.

Note: The run.py script will do the sudo to root internally, which means the `sudo` is not required.

The script contains the commands required to set up the environment:

```
$ cd <PktgenInstallDir>/tools
$ ./run.py -s default  # setup system using the cfg/default.cfg file
```

The run.py script is a python script and tries to configure the system to run a DPDK application. You will probably have to change the configuration files to match your system.

To run pktgen with the default.cfg configuration:

```
$ cd <PktgenInstallDir>/tools
$ run.py default
```

The `run.py` command use python data files to configure setup and run pktgen. The configuration files are located in the `PktgenInstallDir`/cfg directory. These files allow for setup and running pktgen and can be configured to match you system or new configuration files can be created.

Here is the default.cfg file:

```
# Setup configuration
setup = {
 'devices': [
     '81:00.0 81:00.1 81:00.2 81:00.3',
     '85:00.0 85:00.1 85:00.2 85:00.3',
     '83:00.0'
     ],

 'opts': [
     '-b igb_uio'
```

(continues on next page)

(continued from previous page)

```
    ]
 }

# Run command and options
run = {
 'dpdk': [
    '-l 1,1-5,10-13',
    '-n 4',
    '--proc-type auto',
    '--log-level 7',
    '--socket-mem 2048,2048',
    '--file-prefix pg'
    ],

 'blacklist': [
    #'-b 81:00.0 -b 81:00.1 -b 81:00.2 -b 81:00.3',
    #'-b 85:00.0 -b 85:00.1 -b 85:00.2 -b 85:00.3',
    '-b 81:00.0 -b 81:00.1',
    '-b 85:00.0 -b 85:00.1',
    '-b 83:00.0'
    ],

 'pktgen': [
    '-T',
    '-P',
    '--crc-strip',
    '-m [2:3].0',
    '-m [4:5].1',
    '-m [10:11].2',
    '-m [12:13].3',
    ],

 'misc': [
    '-f themes/black-yellow.theme'
   ]
}
```

We have two sections one for setup and the other for running pktgen.

The `modprobe uio` command, in the setup script, loads the UIO support module into the kernel as well as loafing the igb-uio.ko module.

The two echo commands, in the setup script, set up the huge pages for a two socket system. If you only have a single socket system then remove the second echo command. The last command in the script is used to display the hugepage setup.

You may also wish to edit your `.bashrc`, `.profile` or `.cshrc` files to permanently add the environment variables that you set up above:

```
export RTE_SDK=<DPDKInstallDir>
export RTE_TARGET=x86_64-native-linux-gcc
or
export RTE_TARGET=x86_64-native-linux-appgcc
```

1.8 Running the application

Once the above steps have been completed and the `pktgen` application has been compiled you can run it using the commands shown in the *Running Pktgen* section.

CHAPTER 2

Running Pktgen

A sample commandline to start a `pktgen` instance would look something like the following, which you may need 'sudo -E' added to the front if not superuser. The -E option of sudo passes environment variables to sudo shell as the scripts need the RTE_SDK and RTE_TARGET variables:

```
./app/pktgen -l 0-4 -n 3 -- -P -m "[1:3].0, [2:4].1
```

Pktgen, like other DPDK applications splits its commandline arguments into arguments for the DPDK Environmental Abstraction Layer (EAL) and arguments for the application itself. The two sets of arguments are separated using the standard convention of `- -` as shown above.

These commandline arguments are explained in the *EAL Commandline Options* and *Pktgen Commandline Options*.

The output when running `pktgen` will look something like the following:

```
-----------------------

Copyright notices

-----------------------
EAL: Detected lcore 0 as core 0 on socket 0
EAL: Detected lcore 1 as core 1 on socket 0
...
EAL: PCI device 0000:07:00.1 on NUMA socket 0
EAL:   probe driver: 8086:1521 rte_igb_pmd
EAL:   0000:07:00.1 not managed by UIO driver, skipping
Lua 5.2.1  Copyright (C) 1994-2012 Lua.org, PUC-Rio

>>> Packet Burst 16, RX Desc 256, TX Desc 256, mbufs/port 2048, mbuf cache 256

=== port to lcore mapping table (# lcores 5) ===
   lcore:     0     1     2     3     4
port   0:  D: T  1: 0  0: 0  0: 1  0: 0 =  1: 1
port   1:  D: T  0: 0  1: 0  0: 0  0: 1 =  1: 1
Total   :  0: 0  1: 0  1: 0  0: 1  0: 1
```

(continues on next page)

(continued from previous page)

```
    Display and Timer on lcore 0, rx:tx counts per port/lcore

Configuring 6 ports, MBUF Size 1984, MBUF Cache Size 256
Lcore:
    1, type  RX , rx_cnt  1, tx_cnt  0, RX (pid:qid): ( 0: 0) , TX (pid:qid):
    2, type  RX , rx_cnt  1, tx_cnt  0, RX (pid:qid): ( 1: 0) , TX (pid:qid):
    3, type  TX , rx_cnt  0, tx_cnt  1, RX (pid:qid): , TX (pid:qid): ( 0: 0)
    4, type  TX , rx_cnt  0, tx_cnt  1, RX (pid:qid): , TX (pid:qid): ( 1: 0)

Port :
    0, nb_lcores  2, private 0x7d08d8, lcores:  1  3
    1, nb_lcores  2, private 0x7d1c48, lcores:  2  4


Initialize Port 0 -- TxQ 1, RxQ 1,  Src MAC 90:e2:ba:5a:f7:90
    Create: Default RX  0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Default TX  0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Range TX    0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Sequence TX 0:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Special TX  0:0 - Mem (MBUFs   64 x (1984 + 64)) + 790720 =  901 KB

                                                 Port memory used =  20373 KB

Initialize Port 1 -- TxQ 1, RxQ 1,  Src MAC 90:e2:ba:5a:f7:91
    Create: Default RX  1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Default TX  1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Range TX    1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Sequence TX 1:0 - Mem (MBUFs 2048 x (1984 + 64)) + 790720 = 4869 KB
    Create: Special TX  1:0 - Mem (MBUFs   64 x (1984 + 64)) + 790720 =  901 KB

                                                  Port memory used =  20373 KB
                                                Total memory used =  40746 KB

Port  0: Link Up - speed 10000 Mbps - full-duplex <Enable promiscuous mode>
Port  1: Link Up - speed 10000 Mbps - full-duplex <Enable promiscuous mode>


=== Display processing on lcore 0
=== RX processing on lcore  1, rxcnt 1, port/qid, 0/0
=== RX processing on lcore  2, rxcnt 1, port/qid, 1/0
=== TX processing on lcore  3, txcnt 1, port/qid, 0/0
=== TX processing on lcore  4, txcnt 1, port/qid, 1/0
...
```

Once pktgen is running you will see an output like the following:

```
| Ports 0-3 of 8   <Main Page>  Copyright (c) <2010-2019>, Intel Corporation
  Flags:Port      :   P--------------:0   P--------------:1   P--------------:2
 ↪P--------------:3
Link State        :       <UP-10000-FD>       <UP-10000-FD>       <UP-10000-FD>
 ↪    <UP-10000-FD>     ----TotalRate----
Pkts/s Max/Rx     :                 0/0                 0/0                 0/0
 ↪             0/0                   0/0
       Max/Tx     :                 0/0                 0/0                 0/0
 ↪             0/0                   0/0
MBits/s Rx/Tx     :                 0/0                 0/0                 0/0
 ↪             0/0                   0/0
Broadcast         :                   0                   0                   0
 ↪               0
Multicast         :                   0                   0                   0
 ↪               0
```

(continues on next page)

(continued from previous page)

```
  64 Bytes         :                  0                  0                  0
↪                0
  65-127           :                  0                  0                  0
↪                0
  128-255          :                  0                  0                  0
↪                0
  256-511          :                  0                  0                  0
↪                0
  512-1023         :                  0                  0                  0
↪                0
  1024-1518        :                  0                  0                  0
↪                0
Runts/Jumbos       :                0/0                0/0                0/0
↪              0/0
Errors Rx/Tx       :                0/0                0/0                0/0
↪              0/0
Total Rx Pkts      :                  0                  0                  0
↪                0
      Tx Pkts      :                  0                  0                  0
↪                0
      Rx MBs       :                  0                  0                  0
↪                0
      Tx MBs       :                  0                  0                  0
↪                0
ARP/ICMP Pkts      :                0/0                0/0                0/0
↪              0/0
                   :
Pattern Type       :             abcd...            abcd...            abcd...
↪          abcd...
Tx Count/% Rate    :       Forever /100%      Forever /100%      Forever /100%
↪    Forever /100%
PktSize/Tx Burst   :           64 /   32          64 /   32          64 /   32
↪        64 /   32
Src/Dest Port      :         1234 / 5678        1234 / 5678        1234 / 5678
↪      1234 / 5678
Pkt Type:VLAN ID   :     IPv4 / TCP:0001    IPv4 / TCP:0001    IPv4 / TCP:0001
↪  IPv4 / TCP:0001
Dst  IP Address    :         192.168.1.1        192.168.0.1        192.168.3.1
↪      192.168.2.1
Src  IP Address    :      192.168.0.1/24     192.168.1.1/24     192.168.2.1/24
↪   192.168.3.1/24
Dst MAC Address    :   3c:fd:fe:9c:5c:d9   3c:fd:fe:9c:5c:d8   3c:fd:fe:9c:5c:db
↪3c:fd:fe:9c:5c:da
Src MAC Address    :   3c:fd:fe:9c:5c:d8   3c:fd:fe:9c:5c:d9   3c:fd:fe:9c:5c:da
↪3c:fd:fe:9c:5c:db
VendID/PCI Addr    :   8086:1572/04:00.0   8086:1572/04:00.1   8086:1572/04:00.2
↪8086:1572/04:00.3

-- Pktgen Ver: 3.2.4 (DPDK 17.05.0-rc0)  Powered by DPDK ---------------

Pktgen:/>
```

The flags displayed on the top line for each port are:

```
 Flags: P---------------- - Promiscuous mode enabled
        E                 - ICMP Echo enabled
         A                - Send ARP Request flag
          G               - Send Gratuitous ARP flag
           C              - TX Cleanup flag
            p             - PCAP enabled flag
```

(continues on next page)

(continued from previous page)

```
              S           - Send Sequence packets enabled
               R          - Send Range packets enabled
                D         - DPI Scanning enabled (If Enabled)
                 I        - Process packets on input enabled
                  *       - Using TAP interface for this port can be [-rt*]
                   L      - Send Latency packets                          V     -␣
↪Send VLAN ID tag
                    M     - Send MPLS header
                    Q     - Send Q-in-Q tags
                     g    - Process GARP packets
                      g   - Perform GRE with IPv4 payload
                      G   - Perform GRE with Ethernet payload
                       C  - Capture received packets
                        R - Random bitfield(s) are applied

Notes. <state> - Use enable|disable or on|off to set the state.
    <portlist> - a list of ports (no spaces) as 2,4,6-9,12 or 3-5,8 or 5 or the␣
↪word 'all'
         Color best seen on a black background for now
         To see a set of example Lua commands see the files in wr-examples/pktgen/
↪test
```

The pktgen default colors and theme work best on a black background. If required, it is possible to set other color themes, (see commands).

CHAPTER 3

EAL Commandline Options

Pktgen, like other DPDK applications splits commandline arguments into arguments for the DPDK Environmental Abstraction Layer (EAL) and arguments for the application itself. The two sets of arguments are separated using the standard convention of `--`:

Pktgen executable is located at `./app/app/${RTE_TARGET}/pktgen`

pktgen -l 0-4 -n 3 – -P -m “[1:3].0, [2:4].1

The usual EAL commandline usage for `pktgen` is:

```
pktgen -c COREMASK -n NUM \
              [-m NB] \
              [-r NUM] \
              [-b <domain:bus:devid.func>] \
              [--proc-type primary|secondary|auto] -- [pktgen options]
```

The full list of EAL arguments are:

```
EAL options:
  -c COREMASK         : A hexadecimal bitmask of cores to run on
  -n NUM              : Number of memory channels
  -v                  : Display version information on startup
  -d LIB.so           : Add driver (can be used multiple times)
  -m MB               : Memory to allocate (see also --socket-mem)
  -r NUM              : Force number of memory ranks (don't detect)
  --xen-dom0          : Support application running on Xen Domain0 without
                        hugetlbfs
  --syslog            : Set syslog facility
  --socket-mem        : Memory to allocate on specific
                        sockets (use comma separated values)
  --huge-dir          : Directory where hugetlbfs is mounted
  --proc-type         : Type of this process
  --file-prefix       : Prefix for hugepage filenames
  --pci-blacklist, -b : Add a PCI device in black list.
                        Prevent EAL from using this PCI device. The argument
                        format is <domain:bus:devid.func>.
  --pci-whitelist, -w : Add a PCI device in white list.
```

(continues on next page)

(continued from previous page)

```
                        Only use the specified PCI devices. The argument
                        format is <[domain:]bus:devid.func>. This option
                        can be present several times (once per device).
                        NOTE: PCI whitelist cannot be used with -b option
  --vdev              : Add a virtual device.
                        The argument format is <driver><id>[,key=val,...]
                        (ex: --vdev=eth_pcap0,iface=eth2).
  --vmware-tsc-map    : Use VMware TSC map instead of native RDTSC
  --base-virtaddr     : Specify base virtual address
  --vfio-intr         : Specify desired interrupt mode for VFIO
                        (legacy|msi|msix)
  --create-uio-dev    : Create /dev/uioX (usually done by hotplug)

EAL options for DEBUG use only:
  --no-huge           : Use malloc instead of hugetlbfs
  --no-pci            : Disable pci
  --no-hpet           : Disable hpet
  --no-shconf         : No shared config (mmap'd files)
```

The `-c COREMASK` and `-n NUM` arguments are required. The other arguments are optional.

Pktgen requires 2 logical cores (lcore) in order to run. The first lcore, 0, is used for the `pktgen` commandline, for timers and for displaying the runtime metrics text on the terminal. The additional lcores `1-n` are used to do the packet receive and transmits along with anything else related to packets.

You do not need to start at the actual system lcore 0. The application will use the first lcore in the coremask bitmap.

A more typical commandline to start a `pktgen` instance would be:

```
pktgen -l 0-4 -n 3 --proc-type auto --socket-mem 256,256
             -b 0000:03:00.0 -b 0000:03:00.1 \
              --file-prefix pg \
             -- -P -m "[1:3].0, [2:4].1
```

The coremask `-c 0x1f` (0b11111) indicates 5 lcores are used, as the first lcore is used by Pktgen for display and timers.

The `--socket-mem 256,256` DPDK command will allocate 256M from each CPU (two in this case).

The *Pktgen Commandline Options* are shown in the next section.

CHAPTER 4

Pktgen Commandline Options

The Pktgen commandline usage is:

```
./app/app/$(target}/pktgen [EAL options] -- \
                             [-h] [-P] [-G] [-T] [-f cmd_file] \
                             [-l log_file] [-s P:PCAP_file] [-m <string>]
```

The *EAL Commandline Options* were shown in the previous section.

The `pktgen` arguments are:

Usage: ./app/app/x86_64-dnet-linux-gcc/pktgen [EAL options] – [-h] [-P] [-G] [-T] [-f cmd_file] [-l log_fi

-s P:file PCAP packet stream file, 'P' is the port number -f filename Command file (.pkt) to execute or a Lua script (.lua) file -l filename Write log to filename -l use CLI -P Enable PROMISCUOUS mode on all ports -g address Optional IP address and port number default is (localhost:0x5606)

If -g is used that enable socket support as a server application

-G	Enable socket support using default server values localhost:0x5606
-N	Enable NUMA support
-T	Enable the color output
--crc-strip	Strip CRC on all ports
-h	Display the help information

Where the options are:

- `-h`: Display the usage/help information shown above:

```
lspci | grep Ethernet
```

This shows a list of all ports in the system. Some ports may not be usable by DPDK/Pktgen. The first port listed is bit 0 or least signification bit in the `-c` EAL core-

mask. Another method is to compile and run the DPDK sample application `testpmd` to list out the ports DPDK is able to use:

```
./test_pmd -c 0x3 -n 2
```

- `-s P:file`: The PCAP packet file to stream. P is the port number.
- `-f filename`: The script command file (.pkt) to execute or a Lua script (.lua) file. See *Running Script Files*.
- `-l filename`: The filename to write a log to.
- `-P`: Enable PROMISCUOUS mode on all ports.
- `-G`: Enable socket support using default server values of localhost:0x5606. See *Socket Support for Pktgen*.
- `-g address`: Same as `-G` but with an optional IP address and port number. See *Socket Support for Pktgen*.
- `-T`: Enable color terminal output in VT100
- `-N`: Enable NUMA support.
- `-m <string>`: Matrix for mapping ports to logical cores. The format of the port mapping string is defined with a BNF-like grammar as follows:

```
BNF: (or kind of BNF)
<matrix-string>   := """ <lcore-port> { "," <lcore-port>} """
<lcore-port>      := <lcore-list> "." <port-list>
<lcore-list>      := "[" <rx-list> ":" <tx-list> "]"
<port-list>       := "[" <rx-list> ":" <tx-list>"]"
<rx-list>         := <num> { "/" (<num> | <list>) }
<tx-list>         := <num> { "/" (<num> | <list>) }
<list>            := <num> { "/" (<range> | <list>) }
<range>           := <num> "-" <num> { "/" <range> }
<num>             := <digit>+
<digit>           := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

For example:

```
1.0, 2.1, 3.2                - core 1 handles port 0 rx/tx,
                               core 2 handles port 1 rx/tx
                               core 3 handles port 2 rx/tx
1.[0-2], 2.3, ...            - core 1 handle ports 0,1,2 rx/tx,
                               core 2 handle port 3 rx/tx
[0-1].0, [2/4-5].1, ...      - cores 0-1 handle port 0 rx/tx,
                               cores 2,4,5 handle port 1 rx/tx
[1:2].0, [4:6].1, ...        - core 1 handles port 0 rx,
                               core 2 handles port 0 tx,
[1:2].[0-1], [4:6].[2/3], ... - core 1 handles port 0 & 1 rx,
                               core 2 handles port  0 & 1 tx
[1:2-3].0, [4:5-6].1, ...    - core 1 handles port 0 rx, cores 2,3 handle␣
↪port 0 tx
                               core 4 handles port 1 rx & core 5,6 handles␣
↪port 1 tx
[1-2:3].0, [4-5:6].1, ...    - core 1,2 handles port 0 rx, core 3 handles␣
↪port 0 tx
                               core 4,5 handles port 1 rx & core 6 handles␣
↪port 1 tx
[1-2:3-5].0, [4-5:6/8].1, ... - core 1,2 handles port 0 rx, core 3,4,5␣
↪handles port 0 tx
```

(continues on next page)

(continued from previous page)

```
                                core 4,5 handles port 1 rx & core 6,8 handles␣
↪port 1 tx
[1:2].[0:0-7], [3:4].[1:0-7], - core 1 handles port 0 rx, core 2 handles␣
↪ports 0-7 tx
                                core 3 handles port 1 rx & core 4 handles␣
↪port 0-7 tx
BTW: you can use "{}" instead of "[]" as it does not matter to the syntax.
```

Grouping can use { } instead of [ ] if required.

CHAPTER 5

Multiple Instances of Pktgen or other application

One possible solution I use and if you have enough ports available to use. Lets say you need two ports for your application, but you have 4 ports in your system. I physically loop back the cables to have port 0 connect to port 2 and port 1 connected to port 3. Now I can give two ports to my application and two ports to Pktgen.

Setup if pktgen and your application you have to startup each one a bit differently to make sure they share the resources like memory and the ports. I will use two Pktgen running on the same machine, which just means you have to setup your application as one of the applications.

In my machine I have 8 10G ports and 72 lcores between 2 sockets. Plus I have 1024 hugepages per socket for a total of 2048.

Example commands:

```
# lspci | grep Ether
06:00.0 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
06:00.1 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
08:00.0 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
08:00.1 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
09:00.0 Ethernet controller: Intel Corporation I350 Gigabit Network␣
↪Connection (rev 01)
09:00.1 Ethernet controller: Intel Corporation I350 Gigabit Network␣
↪Connection (rev 01)
83:00.1 Ethernet controller: Intel Corporation DH8900CC Null Device (rev␣
↪21)
87:00.0 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
87:00.1 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
89:00.0 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
89:00.1 Ethernet controller: Intel Corporation Ethernet Converged␣
↪Network Adapter X520-Q1 (rev 01)
```

(continues on next page)

(continued from previous page)

```
./app/app/${target}/pktgen -l 2-11 -n 3 --proc-type auto \
           --socket-mem 512,512 --file-prefix pg1 \
           -b 09:00.0 -b 09:00.1 -b 83:00.1 -b 06:00.0 \
           -b 06:00.1 -b 08:00.0 -b 08:00.1 -- \
           -T -P -m "[4:6].0, [5:7].1, [8:10].2, [9:11].3" \
           -f themes/black-yellow.theme

./app/app/${target}/pktgen -l 2,4-11 -n 3 --proc-type auto \
           --socket-mem 512,512 --file-prefix pg2 \
           -b 09:00.0 -b 09:00.1 -b 83:00.1 -b 87:00.0 \
           -b 87:00.1 -b 89:00.0 -b 89:00.1 -- \
           -T -P -m "[12:16].0, [13:17].1, [14:18].2, [15:19].3" \
           -f themes/black-yellow.theme
```

Notice I black list the three onboard devices and then black list the other 4 ports I will not be using for each of the pktgen instances.

I need 8+1 lcores for each instance for Pktgen use. The -c option of ff2 and FF004 lcores, the ff value are used for port handling and the 2/4 is used because pktgen needs the first lcore for display and timers.

The -m option then assigns lcores to the ports.

The information from above is taken from two new files pktgen-master.sh and pktgen-slave.sh, have a look at them and adjust as you need.

Pktgen can also be configured using the commands. `*** Pktgen ***` Copyright &copy <2015-2019>, Intel Corporation.

README for setting up Pktgen with DPDK on Ubuntu 10.04 to 16.10 desktop, it should work on most Linux systems as long as the kernel has hugeTLB page support.

Note: Tested with Ubuntu 13.10 and up to 16.10 kernel versions Linux 3.5.0-25-generic #39-Ubuntu SMP Mon Feb 25 18:26:58 UTC 2013 x86_64

I am using Ubuntu 16.10 x86_64 (64 bit support) for running Pktgen-DPDK on a Crownpass Dual socket board running at 2.4GHz with 32GB of ram 16GB per socket. The current kernel version is 4.4.0-66-generic (as of 2018-04-01) support, but should work on just about any new Linux kernel version.

Currently using as of 2018-04-01 Ubuntu 16.10 Kernel 4.4.0-66-generic system.

To get hugeTLB page support your Linux kernel must be at least 2.6.33 and in the DPDK documents it talks about how you can upgrade your Linux kernel.

Here is another document on how to upgrade your Linux kernel. Ubuntu 10.04 is 2.6.32 by default so upgraded to kernel 2.6.34 using this HOWTO: http://usablesoftware.wordpress.com/2010/05/26/switch-to-a-newer-kernel-in-ubuntu-10-04/

The pktgen output display needs 132 columns and about 42 lines to display currentlyt. I am using an xterm of 132x42, but you can have a larger display and maybe a bit smaller. If you are displaying more then 4-6 ports then you will need a wider display. Pktgen allows you to view a set of ports if they do not all fit on the screen at one time via the ‘page’ command.

Type ‘help’ at the ‘Pktgen>’ prompt to see the complete Pktgen command line commands. Pktgen uses VT100 control codes or escape codes to display the screens, which means your terminal must support VT100. The Hyperterminal in windows is not going to work for Pktgen as it has a few problems with VT100 codes.

Pktgen has a number of modes to send packets single, range, random, sequeue and PCAP modes. Each mode has its own set of packet buffers and you must configure each mode to work correctly. The single packet mode is the information displayed at startup screen or when using the 'page main or page 0' command. The other screens can be accessed using 'page seq|range|rnd|pcap|stats' command.

The pktgen program as built can send up to 16 packets per port in a sequence and you can configure a port using the 'seq' pktgen command. A script file can be loaded from the shell command line via the -f option and you can 'load' a script file from within pktgen as well.

In the BIOS make sure the HPET High Precision Event Timer is enabled. Also make sure hyper-threading is enabled.

** NOTE ** On a 10GB NIC if the transceivers are not attached the screen updates will go very slow.

CHAPTER 6

Pktgen command line directory format

– Pktgen Ver: 3.2.x (DPDK 17.05.0-rc0) Powered by DPDK ————————

Show the commands inside the `pktgen/bin` directory:

```
Pktgen:/> ls
[pktgen]        [sbin]          copyright
Pktgen:/> ls pktgen/bin
off             on              debug           set             pcap
stp             str             stop            start           disable
enable          range           theme           page            seq
sequence        ping4           port            restart         rst
reset           cls             redisplay       save            lua
script          load            geom            geometry        clr
clear.stats     help
Pktgen:/>
```

Showin the `ls` command at root:

```
Pktgen:/> ls
[pktgen]        [sbin]          copyright
Pktgen:/> ls sbin
env             dbg             path            hugepages       cmap
sizes           more            history         quit            clear
pwd             cd              ls              rm              mkdir
chelp           sleep           delay
Pktgen:/>
```

The case of using `ls -l` in a subdirectory:

```
Pktgen:/> cd sbin
Pktgen:/sbin/>
Pktgen:/sbin/> ls -l
  env               Command : Set up environment variables
  dbg               Command : debug commands
  path              Command : display the command path list
  hugepages         Command : hugepages # display hugepage info
  cmap              Command : cmap # display the core mapping
```

(continues on next page)

(continued from previous page)

```
  sizes             Command : sizes # display some internal sizes
  more              Command : more <file> # display a file content
  history           Command : history # display the current history
  quit              Command : quit # quit the application
  clear             Command : clear # clear the screen
  pwd               Command : pwd # display current working directory
  cd                Command : cd <dir> # change working directory
  ls                Command : ls [-lr] <dir> # list current directory
  rm                Command : remove a file or directory
  mkdir             Command : create a directory
  chelp             Command : CLI help - display information for DPDK
  sleep             Command : delay a number of seconds
  delay             Command : delay a number of milliseconds

Pktgen:/sbin/>
Pktgen:/sbin/> cd ..
Pktgen:/>
```

Show help using `ls -l` command in pktgen directory:

```
Pktgen:/pktgen/> cd bin
Pktgen:/pktgen/bin/> ls -l
  off               Alias : disable screen
  on                Alias : enable screen
  debug             Command : debug commands
  set               Command : set a number of options
  pcap              Command : pcap commands
  stp               Alias : stop all
  str               Alias : start all
  stop              Command : stop features
  start             Command : start features
  disable           Command : disable features
  enable            Command : enable features
  range             Command : Range commands
  theme             Command : Set, save, show the theme
  page              Command : change page displays
  seq               Alias : sequence
  sequence          Command : sequence command
  ping4             Command : Send a ping packet for IPv4
  port              Command : Switch between ports
  restart           Command : restart port
  rst               Alias : reset all
  reset             Command : reset pktgen configuration
  cls               Alias : redisplay
  redisplay         Command : redisplay the screen
  save              Command : save the current state
  lua               Command : execute a Lua string
  script            Command : run a Lua script
  load              Command : load command file
  geom              Alias : geometry
  geometry          Command : set the screen geometry
  clr               Alias : clear.stats all
  clear.stats       Command : clear stats
  help              Command : help command

Pktgen:/pktgen/bin/>
```

CHAPTER 7

Runtime Options and Commands

While the pktgen application is running you will see a command prompt as follows:

```
Pktgen:/>
```

From this you can get help or issue runtime commands:

```
Pktgen:/> help

set <portlist> <xxx> value     - Set a few port values
save <path-to-file>            - Save a configuration file using the
                                 filename
load <path-to-file>            - Load a command/script file from the
                                 given path
...
```

The page commands to show different screens:

```
page <pages>                          - Show the port pages or configuration or␣
↪sequence page
    [0-7]                             - Page of different ports
    main                              - Display page zero
    range                             - Display the range packet page
    config | cfg                      - Display the configuration page
    pcap                              - Display the pcap page
    cpu                               - Display some information about the CPU system
    next                              - Display next page of PCAP packets.
    sequence | seq                    - sequence will display a set of packets for a␣
↪given port
                                        Note: use the 'port <number>' to display a new␣
↪port sequence
    rnd                               - Display the random bitfields to packets for a␣
↪given port
                                        Note: use the 'port <number>' to display a new␣
↪port sequence
    log                               - Display the log messages page
    latency                           - Display the latency page
    stats                             - Display physical ports stats for all ports
    xstats                            - Display the extended stats per port
```

List of the `enable/disable` commands:

```
enable|disable <portlist> <features>
    Feature - process               - Enable or Disable processing of ARP/ICMP/IPv4/
↪IPv6 packets
              mpls                  - Enable/disable sending MPLS entry in packets
              qinq                  - Enable/disable sending Q-in-Q header in␣
↪packets
              gre                   - Enable/disable GRE support
              gre_eth               - Enable/disable GRE with Ethernet frame payload
              vlan                  - Enable/disable VLAN tagging
              garp                  - Enable or Disable Gratuitous ARP packet␣
↪processing
              random                - Enable/disable Random packet support
              latency               - Enable/disable latency testing
              pcap                  - Enable or Disable sending pcap packets on a␣
↪portlist
              blink                 - Blink LED on port(s)
              rx_tap                - Enable/Disable RX Tap support
              tx_tap                - Enable/Disable TX Tap support
              icmp                  - Enable/Disable sending ICMP packets
              range                 - Enable or Disable the given portlist for␣
↪sending a range of packets
              capture               - Enable/disable packet capturing on a portlist
              bonding               - Enable call TX with zero packets for bonding␣
↪driver
              short                 - Allow shorter then 64 byte frames to be sent
              vxlan                 - Send VxLAN packets

enable|disable screen               - Enable/disable updating the screen and unlock/
↪lock window
               mac_from_arp         - Enable/disable MAC address from ARP packet
off                                 - screen off shortcut
on                                  - screen on shortcut
```

List of the `set` commands:

```
note: <portlist> - a list of ports (no spaces) e.g. 2,4,6-9,12 or the word 'all'
set <portlist> count <value>        - number of packets to transmit
set <portlist> size <value>         - size of the packet to transmit
set <portlist> rate <percent>       - Packet rate in percentage
set <portlist> burst <value>        - number of packets in a burst
set <portlist> tx_cycles <value>    - DEBUG to set the number of cycles per TX burst
set <portlist> sport <value>        - Source port number for TCP
set <portlist> dport <value>        - Destination port number for TCP
set <portlist> seq_cnt|seqcnt|seqCnt <value>
                                    - Set the number of packet in the sequence to␣
↪send [0-16]
set <portlist> prime <value>        - Set the number of packets to send on prime␣
↪command
set <portlist> dump <value>         - Dump the next N received packets to the screen
set <portlist> vlan <value>         - Set the VLAN ID value for the portlist
set <portlist> jitter <value>       - Set the jitter threshold in micro-seconds
set <portlist> src|dst mac <addr>   - Set MAC addresses 00:11:22:33:44:55 or␣
↪0011:2233:4455 format
set <portlist> type ipv4|ipv6|vlan|arp - Set the packet type to IPv4 or IPv6 or␣
↪VLAN
set <portlist> proto udp|tcp|icmp   - Set the packet protocol to UDP or TCP or ICMP␣
↪per port
set <portlist> pattern <type>       - Set the fill pattern type
                 type - abc         - Default pattern of abc string
                        none        - No fill pattern, maybe random data
```

(continues on next page)

(continued from previous page)

```
                        zero       - Fill of zero bytes
                        user       - User supplied string of max 16 bytes
set <portlist> user pattern <string> - A 16 byte string, must set 'pattern user'␣
↪command
set <portlist> [src|dst] ip ipaddr - Set IP addresses, Source must include network␣
↪mask e.g. 10.1.2.3/24
set <portlist> qinqids <id1> <id2> - Set the Q-in-Q ID's for the portlist
set <portlist> rnd <idx> <off> <mask> - Set random mask for all transmitted␣
↪packets from portlist
    idx: random mask index slot
    off: offset in bytes to apply mask value
    mask: up to 32 bit long mask specification (empty to disable):
          0: bit will be 0
          1: bit will be 1
          .: bit will be ignored (original value is retained)
          X: bit will get random value
set <portlist> cos <value>         - Set the CoS value for the portlist
set <portlist> tos <value>         - Set the ToS value for the portlist
set <portlist> vxlan <flags> <group id> <vxlan_id> - Set the vxlan values
set ports_per_page <value>         - Set ports per page value 1 - 6
```

The range commands:

```
-- Setup the packet range values --
   note: SMMI = start|min|max|inc (start, minimum, maximum, increment)

range <portlist> src|dst mac <SMMI> <etheraddr> - Set destination/source MAC␣
↪address
      e.g: range 0 src mac start 00:00:00:00:00:00
           range 0 dst mac max 00:12:34:56:78:90
      or  range 0 src mac 00:00:00:00:00:00 00:00:00:00:00:00 00:12:34:56:78:90␣
↪00:00:00:01:01:01
range <portlist> src|dst ip <SMMI> <ipaddr>    - Set source IP start address
      e.g: range 0 dst ip start 0.0.0.0
           range 0 dst ip min 0.0.0.0
           range 0 dst ip max 1.2.3.4
           range 0 dst ip inc 0.0.1.0
       or  range 0 dst ip 0.0.0.0 0.0.0.0 1.2.3.4 0.0.1.0
range <portlist> proto tcp|udp                 - Set the IP protocol type
range <portlist> src|dst port <SMMI> <value>   - Set UDP/TCP source/dest port number
       or  range <portlist> src|dst port <start> <min> <max> <inc>
range <portlist> vlan <SMMI> <value>           - Set vlan id start address
      or  range <portlist> vlan <start> <min> <max> <inc>
range <portlist> size <SMMI> <value>           - Set pkt size start address
      or  range <portlist> size <start> <min> <max> <inc>
range <portlist> teid <SMMI> <value>           - Set TEID value
      or  range <portlist> teid <start> <min> <max> <inc>
range <portlist> mpls entry <hex-value>        - Set MPLS entry value
range <portlist> qinq index <val1> <val2>      - Set QinQ index values
range <portlist> gre key <value>               - Set GRE key value
range <portlist> cos <SMMI> <value>            - Set cos value
range <portlist> tos <SMMI> <value>            - Set tos value
```

The sequence commands:

```
sequence <seq#> <portlist> dst <Mac> src <Mac> dst <IP> src <IP> sport <val> dport
↪<val> ipv4|ipv6 udp|tcp|icmp vlan <val> size <val> [teid <val>]
sequence <seq#> <portlist> <dst-Mac> <src-Mac> <dst-IP> <src-IP> <sport> <dport>␣
↪ipv4|ipv6 udp|tcp|icmp <vlanid> <pktsize> [<teid>]
sequence <seq#> <portlist> cos <cos> tos <tos>
```

(continues on next page)

(continued from previous page)

```
    - Set the sequence packet information, make sure the src-IP
      has the netmask value eg 1.2.3.4/24
```

The `pcap` commands:

```
pcap show                           - Show PCAP information
pcap index                          - Move the PCAP file index to the given packet␣
↪number,  0 - rewind, -1 - end of file
pcap filter <portlist> <string>     - PCAP filter string to filter packets on␣
↪receive
```

The `start|stop` commands:

```
start <portlist>                    - Start transmitting packets
stop <portlist>                     - Stop transmitting packets
stp                                 - Stop all ports from transmitting
str                                 - Start all ports transmitting
start <portlist> prime              - Transmit packets on each port listed. See set␣
↪prime command above
start <portlist> arp <type>         - Send a ARP type packet
   type - request | gratuitous | req | grat
```

The debug commands:: dbg l2p - Dump out internal lcore to port mapping dbg tx_dbg - Enable tx debug output dbg mempool <portlist> <type> - Dump out the mempool info for a given type dbg pdump <portlist> - Hex dump the first packet to be sent, single packet mode only dbg memzone - List all of the current memzones dbg memseg - List all of the current memsegs dbg hexdump <addr> <len> - hex dump memory at given address dbg break - break into the debugger dbg memcpy [loop-cnt KBytes] - run a memcpy test

The odd or special commands:

```
save <path-to-file>                 - Save a configuration file using the filename
load <path-to-file>                 - Load a command/script file from the given path
script <filename>                   - Execute the Lua script code in file (www.lua.
↪org).
lua 'lua string'                    - Execute the Lua code in the string needs␣
↪quotes
geometry <geom>                     - Set the display geometry Columns by Rows␣
↪(ColxRow)
clear <portlist> stats              - Clear the statistics
clr                                 - Clear all Statistices
reset <portlist>                    - Reset the configuration the ports to the␣
↪default
rst                                 - Reset the configuration for all ports
ports per page [1-6]                - Set the number of ports displayed per page
port <number>                       - Sets the sequence packets to display for a␣
↪given port
restart <portlist>                  - Restart or stop a ethernet port and restart
ping4 <portlist>                    - Send a IPv4 ICMP echo request on the given␣
↪portlist
```

The `theme` commands:: theme <item> <fg> <bg> <attr> - Set color for item with fg/bg color and attribute value theme show - List the item strings, colors and attributes to the items theme save <filename> - Save the current color theme to a file

Several commands take common arguments such as:

- `portlist`: A list of ports such as `2,4,6-9,12` or the word `all`.

- `state`: This is usually `on` or `off` but will also accept `enable` or `disable`.

For example:

```
Pktgen:/> set all seq_cnt 1
```

The `set` command can also be used to set the MAC address with a format like `00:11:22:33:44:55` or `0011:2233:4455`:

```
set <portlist> src|dst mac etheraddr
```

The `set` command can also be used to set IP addresses:

```
set <portlist> src|dst ip ipaddr
```

7.1 seq

The `seq` command sets the flow parameters for a sequence of packets:

```
seq <seq#> <portlist> dst-Mac src-Mac dst-IP src-IP
                      sport dport ipv4|ipv6|vlan udp|tcp|icmp vid pktsize
```

Where the arguments are:

- `<seq#>`: The packet sequence number.
- `<portlist>`: A portlist as explained above.
- `dst-Mac`: The destination MAC address.
- `src-Mac`: The source MAC address.
- `dst-IP`: The destination IP address.
- `src-IP`: The source IP address. Make sure the src-IP has the netmask value such as `1.2.3.4/24`.
- `sport`: The source port.
- `dport`: The destination port.
- `IP`: The IP layer. One of `ipv4|ipv6|vlan`.
- `Transport`: The transport. One of `udp|tcp|icmp`.
- `vid`: The VLAN ID.
- `pktsize`: The packet size.

7.2 save

The `save` command saves the current configuration of a file:

```
save <path-to-file>
```

7.3 load

The `load` command loads a configuration from a file:

```
load <path-to-file>
```

The is most often used with a configuration file written with the `save` command, see above.

7.4 ports per page

The `ports per page` (ports per page) command sets the number of ports displayed per page:

```
ports per page [1-6]
```

7.5 script

The `script` command execute the Lua code in specified file:

```
script <filename>
```

See *Running Script Files*.

7.6 pages

The Random or rnd page.

```
Port 0           <Random bitfield Page>  Copyright (c) <2010-2019>, Intel␣
↪Corporation
  Index   Offset     Act?  Mask [0 = 0 bit, 1 = 1 bit, X = random bit, . = ignore]
     0        0      No   00000000 00000000 00000000 00000000
     1        0      No   00000000 00000000 00000000 00000000
     2        0      No   00000000 00000000 00000000 00000000
     3        0      No   00000000 00000000 00000000 00000000
     4        0      No   00000000 00000000 00000000 00000000
     5        0      No   00000000 00000000 00000000 00000000
     6        0      No   00000000 00000000 00000000 00000000
     7        0      No   00000000 00000000 00000000 00000000
     8        0      No   00000000 00000000 00000000 00000000
     9        0      No   00000000 00000000 00000000 00000000
     10       0      No   00000000 00000000 00000000 00000000
     11       0      No   00000000 00000000 00000000 00000000
     12       0      No   00000000 00000000 00000000 00000000
     13       0      No   00000000 00000000 00000000 00000000
     14       0      No   00000000 00000000 00000000 00000000
     15       0      No   00000000 00000000 00000000 00000000
     16       0      No   00000000 00000000 00000000 00000000
     17       0      No   00000000 00000000 00000000 00000000
     18       0      No   00000000 00000000 00000000 00000000
     19       0      No   00000000 00000000 00000000 00000000
     20       0      No   00000000 00000000 00000000 00000000
     21       0      No   00000000 00000000 00000000 00000000
```

(continues on next page)

(continued from previous page)

```
     22       0       No   00000000 00000000 00000000 00000000
     23       0       No   00000000 00000000 00000000 00000000
     24       0       No   00000000 00000000 00000000 00000000
     25       0       No   00000000 00000000 00000000 00000000
     26       0       No   00000000 00000000 00000000 00000000
     27       0       No   00000000 00000000 00000000 00000000
     28       0       No   00000000 00000000 00000000 00000000
     29       0       No   00000000 00000000 00000000 00000000
     30       0       No   00000000 00000000 00000000 00000000
     31       0       No   00000000 00000000 00000000 00000000
     -- Pktgen Ver: 3.2.4 (DPDK 17.05.0-rc0)  Powered by DPDK -----
```

The sequence or seq page.

```
<Sequence Page>  Copyright (c) <2010-2019>, Intel Corporation
  Port   :  0, Sequence Count:  8 of 16                                        ␣
↪                                 GTPu
    * Seq:            Dst MAC           Src MAC          Dst IP            Src IP ␣
↪   Port S/D Protocol:VLAN  Size  TEID
    *   0:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0
    *   1:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0
    *   2:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0
    *   3:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0
    *   4:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0
    *   5:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0
    *   6:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0
    *   7:  3c:fd:fe:9c:5c:d9 3c:fd:fe:9c:5c:d8     192.168.1.1    192.168.0.1/24 ␣
↪ 1234/5678 IPv4/TCP:0001   64     0

    -- Pktgen Ver: 3.2.4 (DPDK 17.05.0-rc0)  Powered by DPDK ---------------
```

The CPU information page.

```
<CPU Page>  Copyright (c) <2010-2019>, Intel Corporation

Kernel: Linux rkwiles-DESK1.intel.com 4.4.0-66-generic #87-Ubuntu SMP Fri Mar 3␣
↪15:29:05 UTC 2018 x86_64 x86_64 x86_64 GNU/Linux

Model Name: Intel(R) Xeon(R) CPU E5-2699 v3 @ 2.30GHz
CPU Speed : 1201.031
Cache Size: 46080 KB

CPU Flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat␣
↪pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp␣
↪lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc␣
↪aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3␣
↪sdbg fma cx16 xtpr pdcm pcid dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_
↪timer aes xsave avx f16c rdrand lahf_lm abm epb tpr_shadow vnmi flexpriority ept␣
↪vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid cqm xsaveopt cqm_llc␣
↪cqm_occup_llc dtherm ida arat pln pts
  2 sockets, 18 cores per socket and 2 threads per core.
  Socket   :    0         1
  Core   0 : [ 0,36]   [18,54]
```

(continues on next page)

(continued from previous page)

```
  Core   1 : [ 1,37]   [19,55]
  Core   2 : [ 2,38]   [20,56]
  Core   3 : [ 3,39]   [21,57]
  Core   4 : [ 4,40]   [22,58]
  Core   5 : [ 5,41]   [23,59]
  Core   6 : [ 6,42]   [24,60]
  Core   7 : [ 7,43]   [25,61]
  Core   8 : [ 8,44]   [26,62]
  Core   9 : [ 9,45]   [27,63]
  Core  10 : [10,46]   [28,64]
  Core  11 : [11,47]   [29,65]
  Core  12 : [12,48]   [30,66]
  Core  13 : [13,49]   [31,67]
  Core  14 : [14,50]   [32,68]
  Core  15 : [15,51]   [33,69]
  Core  16 : [16,52]   [34,70]
  Core  17 : [17,53]   [35,71]
```

The latency page.

```
-- Ports 0-3 of 8   <Main Page>  Copyright (c) <2010-2019>, Intel Corporation
       Flags:Port        :   P----S---------:0   P--------------:1   P-----------
↪--:2   P--------------:3
       Link State        :       <UP-10000-FD>       <UP-10000-FD>       <UP-
↪10000-FD>       <UP-10000-FD>     ----TotalRate----
       Pkts/s Max/Rx     :                 0/0                 0/0
↪ 0/0                 0/0                  0/0
              Max/Tx     :                 0/0                 0/0
↪ 0/0                 0/0                  0/0
       MBits/s Rx/Tx     :                 0/0                 0/0
↪ 0/0                 0/0                  0/0
                         :
       Latency usec      :                   0                   0
↪   0                   0
       Jitter Threshold  :                  50                  50
↪  50                  50
       Jitter count      :                   0                   0
↪   0                   0
       Total Rx pkts     :                   0                   0
↪   0                   0
       Jitter percent    :                   0                   0
↪   0                   0
                         :
       Pattern Type      :             abcd...             abcd...
↪abcd...             abcd...
       Tx Count/% Rate   :       Forever /100%       Forever /100%       Forever /
↪100%       Forever /100%
       PktSize/Tx Burst  :           64 /   32           64 /   32           64 /
↪  32           64 /   32
       Src/Dest Port     :         1234 / 5678         1234 / 5678         1234 /
↪5678         1234 / 5678
       Pkt Type:VLAN ID  :     IPv4 / TCP:0001     IPv4 / TCP:0001     IPv4 /
↪TCP:0001     IPv4 / TCP:0001
       Dst  IP Address   :         192.168.1.1         192.168.0.1         192.
↪168.3.1         192.168.2.1
       Src  IP Address   :      192.168.0.1/24      192.168.1.1/24      192.168.2.
↪1/24      192.168.3.1/24
       Dst MAC Address   :   3c:fd:fe:9c:5c:d9   3c:fd:fe:9c:5c:d8
↪3c:fd:fe:9c:5c:db   3c:fd:fe:9c:5c:da
       Src MAC Address   :   3c:fd:fe:9c:5c:d8   3c:fd:fe:9c:5c:d9
↪3c:fd:fe:9c:5c:da   3c:fd:fe:9c:5c:db
```

(continues on next page)

(continued from previous page)

```
       VendID/PCI Addr   :   8086:1572/04:00.0   8086:1572/04:00.1   8086:1572/
↪04:00.2   8086:1572/04:00.3

       -- Pktgen Ver: 3.2.4 (DPDK 17.05.0-rc0)  Powered by DPDK ---------------
```

The config or cfg page.

```
<CPU Page>  Copyright (c) <2010-2019>, Intel Corporation
 2 sockets, 18 cores, 2 threads
  Socket   :    0         1      Port description
  Core   0 : [ 0,36]   [18,54]   0000:04:00.0 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core   1 : [ 1,37]   [19,55]   0000:04:00.1 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core   2 : [ 2,38]   [20,56]   0000:04:00.2 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core   3 : [ 3,39]   [21,57]   0000:04:00.3 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core   4 : [ 4,40]   [22,58]   0000:05:00.0 : Intel Corporation I350 Gigabit␣
↪Network Connection (rev 01)
  Core   5 : [ 5,41]   [23,59]   0000:05:00.1 : Intel Corporation I350 Gigabit␣
↪Network Connection (rev 01)
  Core   6 : [ 6,42]   [24,60]   0000:81:00.0 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core   7 : [ 7,43]   [25,61]   0000:81:00.1 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core   8 : [ 8,44]   [26,62]   0000:81:00.2 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core   9 : [ 9,45]   [27,63]   0000:81:00.3 : Intel Corporation X710 for 10GbE␣
↪SFP+ (rev 01)
  Core  10 : [10,46]   [28,64]   0000:82:00.0 : Intel Corporation XL710 for 40GbE␣
↪QSFP+ (rev 02)
  Core  11 : [11,47]   [29,65]   0000:83:00.0 : Intel Corporation XL710 for 40GbE␣
↪QSFP+ (rev 02)
  Core  12 : [12,48]   [30,66]
  Core  13 : [13,49]   [31,67]
  Core  14 : [14,50]   [32,68]
  Core  15 : [15,51]   [33,69]
  Core  16 : [16,52]   [34,70]
  Core  17 : [17,53]   [35,71]

  -- Pktgen Ver: 3.2.4 (DPDK 17.05.0-rc0)  Powered by DPDK ---------------
```

Here is the page range screen.

```
Port #                          Port-0            Port-1            Port-2  ␣
↪          Port-3
dst.ip                :       192.168.1.1       192.168.2.1       192.168.3.1  ␣
↪     192.168.4.1
    inc               :             0.0.0.1           0.0.0.1           0.0.0.1  ␣
↪         0.0.0.1
    min               :       192.168.1.1       192.168.2.1       192.168.3.1  ␣
↪     192.168.4.1
    max               :     192.168.1.254     192.168.2.254     192.168.3.254  ␣
↪   192.168.4.254
                      :
src.ip                :       192.168.0.1       192.168.1.1       192.168.2.1  ␣
↪     192.168.3.1
    inc               :             0.0.0.0           0.0.0.0           0.0.0.0  ␣
↪         0.0.0.0
```

(continues on next page)

(continued from previous page)

```
    min           :          192.168.0.1          192.168.1.1          192.168.2.1
↪     192.168.3.1
    max           :        192.168.0.254        192.168.1.254        192.168.2.254
↪   192.168.3.254
                  :
ip_proto          :                  TCP                  TCP                  TCP
↪             TCP
                  :
dst.port / inc    :              0/    1              256/    1              512/    1
↪       768/    1
     min / max    :              0/  254              256/  510              512/  766
↪       768/ 1022
                  :
src.port / inc    :              0/    1              256/    1              512/    1
↪       768/    1
     min / max    :              0/  254              256/  510              512/  766
↪       768/ 1022
                  :
vlan.id / inc     :               1/   0               1/   0               1/   0
↪           1/   0
    min / max     :              1/4095               1/4095               1/4095
↪          1/4095
                  :
pkt.size / inc    :              64/   0              64/   0              64/   0
↪         64/   0
     min / max    :              64/1518              64/1518              64/1518
↪         64/1518
                  :
dst.mac           :   3c:fd:fe:9c:5c:d9   3c:fd:fe:9c:5c:d8   3c:fd:fe:9c:5c:db
↪3c:fd:fe:9c:5c:da
    inc           :   00:00:00:00:00:00   00:00:00:00:00:00   00:00:00:00:00:00
↪00:00:00:00:00:00
    min           :   00:00:00:00:00:00   00:00:00:00:00:00   00:00:00:00:00:00
↪00:00:00:00:00:00
    max           :   00:00:00:00:00:00   00:00:00:00:00:00   00:00:00:00:00:00
↪00:00:00:00:00:00
                  :
src.mac           :   3c:fd:fe:9c:5c:d8   3c:fd:fe:9c:5c:d9   3c:fd:fe:9c:5c:da
↪3c:fd:fe:9c:5c:db
    inc           :   00:00:00:00:00:00   00:00:00:00:00:00   00:00:00:00:00:00
↪00:00:00:00:00:00
    min           :   00:00:00:00:00:00   00:00:00:00:00:00   00:00:00:00:00:00
↪00:00:00:00:00:00
    max           :   00:00:00:00:00:00   00:00:00:00:00:00   00:00:00:00:00:00
↪00:00:00:00:00:00
                  :
gtpu.teid / inc   :              0/    0              0/    0              0/    0
↪         0/    0
      min / max   :              0/    0              0/    0              0/    0
↪         0/    0
-- Pktgen Ver: 3.2.4 (DPDK 17.05.0-rc0)  Powered by DPDK ---------------

Pktgen:/>
```

s

CHAPTER 8

CLI Sample Application

CLI stands for “Command Line Interface”.

This chapter describes the CLI sample application that is part of the Data Plane Development Kit (DPDK). The CLI is a workalike replacement for cmdline library in DPDK and has a simpler programming interface and programming model.

The primary goal of CLI is to allow the developer to create commands quickly and with very little compile or runtime configuration. Using standard Unix* like constructs which are very familar to the developer. Allowing the developer to construct a set of commands for development or deployment of the application.

The CLI design uses a directory like design instead of a single level command line interface. Allowing the developer to use a directory style solution to controlling a DPDK application. The directory style design is nothing new, but it does have some advantages.

One advantage allows the directory path for the command to be part of the information used in executing the command. The next advantage is creating directories to make a hierarchy of commands, plus allowing whole directroy trees to dynamically come and go as required by the developer.

Some of the advantages are:

- CLI has no global variable other then the single thread variable called *this_cli* which can only be accessed from the thread which created the CLI instance.
- **CLI supports commands, files, aliases, directories.**
  - The alias command is just a string using a simple substitution support for other commands similar to the bash shell like alias commands.
  - Files can be static or dynamic information, can be changed on the fly and saved for later. The file is backed with a simple function callback to allow the developer to update the content or not.
- Added support for color and cursor movement APIs similar to Pktgen if needed by the developer.

- It is a work alike replacement for cmdline library. Both cmdline and CLI can be used in the same application if care is taken.
- Uses a simple fake like directory layout for command and files. Allowing for command hierarchy as path to the command can allow for specific targets to be identified without having to state it on the command line.
- Has auto-complete for commands, similar to Unix/Linux autocomplete and provides support for command option help as well.
- **Callback functions for commands are simply just argc/argv like functions.**
  - The CLI does not convert arguments for the user, it is up to the developer to decode the argv[] values.
  - Most of the arguments converted in the current cmdline are difficult to use or not required as the developer just picks string type and does the conversion himself.
- Dynamically be able to add and remove commands, directories, files and aliases, does not need to be statically compiled into the application.
- No weird structures in the code and reduces the line count for testpmd from 12K to 4.5K lines. I convert testpmd to have both CMDLINE and CLI with a command line option.
- **Two methods to parse command lines, first is the standard argc/argv method in the function.**
  - The second method is to use a map of strings with simple printf like formatting to detect which command line the user typed.
  - An ID value it returned to the used to indicate which mapping string was found to make the command line to be used in a switch statement.
- Environment variable support using the **env** command or using an API.
- Central help support if needed (optional).

8.1 Overview

The CLI sample application is a simple application that demonstrates the use of the command line interface in the DPDK. This application is a readline-like interface that can be used to control a DPDK application.

One of the advantages of CLI over Cmdline is it is dynamic, which means nodes or items can be added and removed on the fly. Which allows adding new directories, file or commands as needed or removing these items at runtime. The CLI has no global modifiable variable as the one global pointer is a thread based variable. Which allows the developer to have multiple CLI commands per thread if needed.

Another advantage is the calling of the backend function to support a command is very familar to developers as it is basically just a argc/argv style command and the developer gets the complete command line.

One other advantage is the use of MAP structures, to help identify commands quickly plus allowing the developer to define new versions of commands and be able to identify these new versions using a simple identifier value. Look at the sample application to see a simple usage.

Another advantage of CLI is how simple it is to add new directroies, files and commands for user development. The basic concept is for the developer to use standard Unix like designs. To add a command a developer needs to add an entry to the cli_tree_t structure and create a function using the following prototype:

```
int user_cmd(int argc, char **argv);
```

The argc/argv is exactly like the standard usage in a Unix* system, which allows for using getopt() and other standard functions. The Cmdline structures and text conversions were defined at compile time in most cases. In CLI the routine is passed the argc/argv information to convert these options as needed. The cli variable being a thread Local Storage (TLS) all user routines a CLI routine only need to access the thread variable to eliminate needing a global variable to reference the specific CLI instance and passing the value in the API.

The user can also set environment variables using the **env** command. These variables are also parsed in the command line a direct substitution is done.

The CLI system also has support for simple files along with alias like commands. These alias commands are fixed strings which are executed instead of a function provided by the developer. If the user has more arguments these are appended to the alias string and processed as if typed on the command line.

Note: The CLI library was designed to be used in production code and the Cmdline was not validated to the same standard as other DPDK libraries. The goal is to provide a production CLI design.

The CLI sample application supports some of the features of the Cmdline library such as, completion, cut/paste and some other special bindings that make configuration and debug faster and easier.

The CLI design uses some very simple VT100 control strings for displaying data and accepting input from the user. Some of the control strings are used to clear the screen or line and position the cursor on a VT100 compatible terminal. The CLI screen code also supports basic color and many other VT100 commands.

The application also shows how the CLI application can be extended to handle a list of commands and user input.

The example presents a simple command prompt **DPDK-cli:/>** similar to a Unix* shell command along with a directory like file system.

Some of the **default** commands contained under /sbin directory are:

- **ls**: list the current or provided directory files/commands.
- **cd**: Change directory command.
- **pwd**: print out the current working directory.
- **history**: List the current command line history if enabled.
- **more**: A simple command to page contents of files.
- **help**: display a the help screen.
- **quit**: exit the CLI application, also **Ctrl-x** will exit as well.
- **mkdir**: add a directory to the current directory.

- **delay**: wait for a given number of microseconds.
- **sleep**: wait for a given number of seconds.
- **rm**: remove a directory, file or command. Removing a file will delete the data.
- **cls**: clear the screen and redisplay the prompt.
- **version**: Display the current DPDK version being used.
- **path**: display the current search path for executable commands.
- **cmap**: Display the current system core and socket information.
- **hugepages**: Display the current hugepage information.
- **sizes**: a collection system structure and buffer sizes for debugging.
- **copyright**: a file containing DPDK copyright information.
- **env**: a command show/set/modify the environment variables.

Some example commands under /bin directory are:

- **ll**: an alias command to display long ls listing **ls -l**
- **h**: alias command for **history**
- **hello**: a simple Hello World! command.
- **show**: has a number of commands using the map feature.

Under the /data directory is:

- **pci**: a simple example file for displaying the **lspci** command in CLI.

Note: To terminate the application, use **Ctrl-x** or the command **quit**.

8.2 Auto completion

CLI does support auto completion at the file or directory level, meaning the arguments to commands are not expanded as was done in Cmdline code. The CLI auto completion works similar to the standard Unix* system by expanding commands and directory paths. In normal Unix* like commands the user needs to execute the command asking for the help information and CLI uses this method.

8.3 Special command features

Using the '!' followed by a number from the history list of commands you can execute that command again. Using the UP/Down arrows the user can quickly find and execute or modify a previous command in history.

The user can also execute host level commands if enabled using the '@' prefix to a command line e.g. @ls or @lspci or ... line is passed to popen or system function to be executed and the output displayed on the console if any output.

8.4 Compiling the Application

1. Go to example directory:

```
export RTE_SDK=/path/to/rte_sdk
cd ${RTE_SDK}/examples/cli
```

1. Set the target (a default target is used if not specified). For example:

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

Refer to the *DPDK Getting Started Guide* for possible RTE_TARGET values.
```

1. Build the application:

```
make
```

8.5 Running the Application

To run the application in linux environment, issue the following command:

```
$ ./build/cli
```

Note: The example cli application does not require to be run as superuser as it does not startup DPDK by calling rte_eal_init() routine. Which means it also does not use DPDK features except for a few routines not requiring EAL initialization.

Refer to the *DPDK Getting Started Guide* for general information on running applications and the Environment Abstraction Layer (EAL) options.

8.6 Explanation

The following sections provide some explanation of the code.

8.6.1 EAL Initialization and cmdline Start

The first task is the initialization of the Environment Abstraction Layer (EAL), if required for the application.

```
int
main(int argc, char **argv)
{
    if (cli_create_with_tree(init_tree) ==0) {
        cli_start(NULL, 0); /* NULL is some init message done only once */
                            /* 0 means do not use color themes */
        cli_destroy();
    }
    ...
```

The cli_start() function returns when the user types **Ctrl-x** or uses the quit command in this case, the application exits. The cli_create() call takes four arguments and each has a default value if not provided. The API used here is the cli_create_with_tree(), which uses defaults for three of the arguments.

```
/**
* Create the CLI engine
*
* @param prompt_func
*   Function pointer to call for displaying the prompt.
* @param tree_func
*   The user supplied function to init the tree or can be NULL. If NULL then
*   a default tree is initialized with basic commands.
* @param nb_entries
*   Total number of commands, files, aliases and directories. If 0 then use
*   the default number of nodes. If -1 then unlimited number of nodes.
* @param nb_hist
*   The number of lines to keep in history. If zero then turn off history.
*   If the value is CLI_DEFAULT_HISTORY use CLI_DEFAULT_HIST_LINES
* @return
*   0 on success or -1
*/
int cli_create(cli_prompt_t prompt_func, cli_tree_t tree_func,
                    int nb_entries, uint32_t nb_hist);
```

The cli_create_with_tree() has only one argument which is the structure to use in order to setup the initial directory structure. Also the wrapper function int cli_create_with_defaults(void) can be used as well.

Consult the cli.h header file for the default values. Also the alias node is a special alias file to allow for aliasing a command to another command.

The tree init routine is defined like:

```
static struct cli_tree my_tree[] = {
    c_dir("/data"),
    c_file("pci", pci_file, "display lspci information"),
    c_dir("/bin"),
    c_cmd("hello", hello_cmd, "Hello-World!!"),
    c_alias("h", "history", "display history commands"),
    c_alias("ll", "ls -l", "long directory listing alias"),
    c_end()
};

static int
init_tree(void)
{
    /*
     * Root is created already and using system default cmds and dirs, the
     * developer is not required to use the system default cmds/dirs.
     */
    if (cli_default_tree_init())
        return -1;

        /* Using NULL here to start at root directory */
    if (cli_add_tree(NULL, my_tree))
        return -1;

        cli_help_add("Show", show_map, show_help);

        return cli_add_bin_path("/bin");
```

(continues on next page)

(continued from previous page)

```
}
```

The above structure is used to create the tree structure at initialization time. The struct cli_tree or cli_tree_t typedef can be used to setup a new directory tree or argument the default tree.

The elements are using a set of macros c_dir, c_file, c_cmd, c_alias and c_end. These macros help fill out the cli_tree_t structure for the given type of item.

The developer can create his own tree structure with any commands that are needed and/or call the cli_default_tree_init() routine to get the default structure of commands. If the developer does not wish to call the default CLI routine, then he must call the cli_create_root() function first before adding other nodes. Other nodes can be added and removed at anytime.

8.6.2 CLI Map command support

The CLI command has two types of support to handle arguments normal argc/argv and the map system. As shown above the developer creates a directory tree and attaches a function to a command. The function takes the CLI pointer plus the argc/argv arguments and the developer can just parse the arguments to decode the command arguments. Sometimes you have multiple commands or different versions of a command being handled by a single routine, this is were the map support comes into play.

The map support defines a set of struct cli_map map[]; to help detect the correct command from the user. In the list of cli_map structures a single structure contains two items a developer defined index value and a command strings. The index value is used on the function to identify the specific type of command found in the list. The string is a special printf like string to help identify the command typed by the user. One of the first things todo in the command routine is to call the cli_mapping() function passing in the CLI pointer and the argc/argv values.The two method can be used at the same time.

The cli_mapping() command matches up the special format string with the values in the argc/argv array and returns the developer supplied index value or really the pointer the struct cli_map instance.

Now the developer can use the cli_map.index value in a switch() statement to locate the command the user typed or if not found a return of -1.

Example:

```
static int
hello_cmd(int argc, char **argv)
{
    int i, opt;

    optind = 1;
    while((opt = getopt(argc, argv, "?")) != -1) {
        switch(opt) {
            case '?': cli_usage(); return 0;
            default:
                break;
        }
    }

    cli_printf("Hello command said: Hello World!! ");
    for(i = 1; i < argc; i++)
```

(continues on next page)

(continued from previous page)

```
        cli_printf("%s ", argv[i]);
    cli_printf("\n");

    return 0;
}

static int
pci_file(struct cli_node *node, char *buff, int len, uint32_t opt)
{
        if (is_file_open(opt)) {
                FILE *f;

                if (node->file_data && (node->fflags & CLI_FREE_DATA))
                        free(node->file_data);

                node->file_data = malloc(32 * 1024);
                if (!node->file_data)
                        return -1;
                node->file_size = 32 * 1024;
                node->fflags = CLI_DATA_RDONLY | CLI_FREE_DATA;

                f = popen("lspci", "r");
                if (!f)
                        return -1;

                node->file_size = fread(node->file_data, 1, node->file_size, f);

                pclose(f);
        return 0;
    }
    return cli_file_handler(node, buff, len, opt);
}

static struct cli_map show_map[] = {
        { 10, "show %P" },
        { 20, "show %P mac %m" },
        { 30, "show %P vlan %d mac %m" },
        { 40, "show %P %|vlan|mac" },
        { -1, NULL }
};

static const char *show_help[] = {
        "show <portlist>",
        "show <portlist> mac <pg_ether_addr>",
        "show <portlist> vlan <vlanid> mac <pg_ether_addr>",
        "show <portlist> [vlan|mac]",
        NULL
};

static int
show_cmd(int argc, char **argv)
{
        struct cli_map *m;
        uint32_t portlist;
        struct pg_ether_addr mac;

        m = cli_mapping(Show_info.map, argc, argv);
        if (!m)
                return -1;

        switch(m->index) {
```

(continues on next page)

(continued from previous page)

```
                case 10:
                        portlist_parse(argv[1], &portlist);
                        cli_printf("   Show Portlist: %08x\n", portlist);
                        break;
                case 20:
                        portlist_parse(argv[1], &portlist);
                        pg_ether_aton(argv[3], &mac);
                        cli_printf("   Show Portlist: %08x, MAC: "
                                   "%02x:%02x:%02x:%02x:%02x:%02x\n",
                                      portlist,
                                      mac.addr_bytes[0],
                                      mac.addr_bytes[1],
                                      mac.addr_bytes[2],
                                      mac.addr_bytes[3],
                                      mac.addr_bytes[4],
                                      mac.addr_bytes[5]);
                        break;
                case 30:
                        portlist_parse(argv[1], &portlist);
                        pg_ether_aton(argv[5], &mac);
                        cli_printf("   Show Portlist: %08x vlan %d MAC: "
                                   "%02x:%02x:%02x:%02x:%02x:%02x\n",
                                      portlist,
                                      atoi(argv[3]),
                                      mac.addr_bytes[0],
                                      mac.addr_bytes[1],
                                      mac.addr_bytes[2],
                                      mac.addr_bytes[3],
                                      mac.addr_bytes[4],
                                      mac.addr_bytes[5]);
                        break;
                case 40:
                        portlist_parse(argv[1], &portlist);
                        pg_ether_aton("1234:4567:8901", &mac);
                        cli_printf("   Show Portlist: %08x %s: ",
                                      portlist, argv[2]);
                        if (argv[2][0] == 'm')
                                cli_printf("%02x:%02x:%02x:%02x:%02x:%02x\n",
                                      mac.addr_bytes[0],
                                      mac.addr_bytes[1],
                                      mac.addr_bytes[2],
                                      mac.addr_bytes[3],
                                      mac.addr_bytes[4],
                                      mac.addr_bytes[5]);
                        else
                                cli_printf("%d\n", 101);
                        break;
                default:
                        cli_help_show_group("Show");
                        return -1;
        }
        return 0;
}

static struct cli_tree my_tree[] = {
        c_dir("/data"),
        c_file("pci",   pci_file,       "display lspci information"),
        c_dir("/bin"),
        c_cmd("show",   show_cmd,       "show mapping options"),
        c_cmd("hello",  hello_cmd,      "Hello-World!!"),
        c_alias("h",    "history",      "display history commands"),
```

(continues on next page)

(continued from previous page)

```
        c_alias("ll",   "ls -l",        "long directory listing alias"),
        c_end()
};
```

Here is the cli_tree for this example, note it has a lot more commands. The show_cmd or show command is located a number of lines down. This cli_tree creates in the /bin directory a number of commands, which one is the show command. The show command has four different formats if you look at the show_map[].

The user types one of these commands and cli_mapping() attempts to locate the correct entry in the list. You will also notice another structure called pcap_help, which is an array of strings giving a cleaner and longer help description of each of the commands.

These two structure show_map/show_help can be added to the cli_help system to provide help for a command using a simple API.

or we can use the cli_help_show_all() API to show all added help information.

```
cli_help_show_all(NULL);
```

The following is from Pktgen source code to add more help to the global help for the system.

```
cli_help_add("Title", NULL, title_help);
cli_help_add("Page", page_map, page_help);
cli_help_add("Enable", enable_map, enable_help);
cli_help_add("Set", set_map, set_help);
cli_help_add("Range", range_map, range_help);
cli_help_add("Sequence", seq_map, seq_help);
cli_help_add("PCAP", pcap_map, pcap_help);
cli_help_add("Start", start_map, start_help);
cli_help_add("Debug", debug_map, debug_help);
cli_help_add("Misc", misc_map, misc_help);
cli_help_add("Theme", theme_map, theme_help);
cli_help_add("Status", NULL, status_help);
```

8.6.3 Understanding the CLI system

The command line interface is defined as a fake directory tree with executables, directories and files. The user uses shell like standard commands to move about the directory and execute commands. The CLI is not a powerful as the Bash shell, but has a number of similar concepts.

Our fake directory tree has a '/' or root directory which is created when cli_create() is called along with the default sbin directory. The user starts out at the root directory '/' and is allowed to cd to other directories, which could contain more executables, aliases or directories. The max number of directory levels is limited to the number of nodes given at startup.

The default directory tree starts out as just root (/) and a sbin directory. Also it contains a file called copyright in root, which can be displayed using the default 'more copyright' command.

A number of default commands are predefined in the /sbin directory and are defined above. Other bin directories can be added to the system if needed, but a limit of CLI_MAX_BINS is defined in the cli.h header file.

The CLI structure is created at run time adding directories, commands and aliases as needed, which is different from the cmdline interface in DPDK today.

The basic concept for a command is similar to a standard Linux executable, meaning the command when executed it is passed the command line in a argc/argv format to be parsed by the function. The function is attached to a command file in the directory tree and is executed when the user types the name of the function along with it arguments. Some examples of the default commands can be seen in the lib/librte_cli/cli_cmds.c file.

CHAPTER 9

CLI library guide

CLI stands for “Command Line Interface”.

This chapter describes the CLI library which is a part of the Data Plane Development Kit (DPDK). The CLI is a workalike replacement for cmdline library in DPDK and has a simpler interface and programming model plus it is dynamic.

The primary goal of CLI is to allow the developer to create commands quickly and with very little compile or runtime configuration. Using standard Unix* like constructs which are very familar to the developer. Allowing the developer to construct a set of commands for development or deployment of the application.

The CLI design uses a directory like design instead of a single level command line interface. Allowing the developer to use a directory style solution to controlling a DPDK application. The directory style design is nothing new, but it does have some advantages over a single level command structure.

One advantage allows the directory path for the command to be part of the information used in executing the command. The next advantage is creating directories to make a hierarchy of commands, plus allowing whole directroy trees to dynamicly come and go as required by the developer.

Some of the advantages are:

- CLI has no global variable other then the single thread variable called *this_cli* which can only be accessed from the thread which created the CLI instance.
- **CLI supports commands, files, aliases, directories.**
  - The alias command is just a string using a simple substitution support for other commands similar to the bash shell like alias commands.
  - Files can be static or dynamic information, can be changed on the fly and saved for later. The file is backed with a simple function callback to allow the developer to update the content or not.
- Added support for color and cursor movement APIs similar to Pktgen if needed by the developer.

- It is a work alike replacement for cmdline library. Both cmdline and CLI can be used in the same application if care is taken.
- Uses a simple fake like directory layout for command and files. Allowing for command hierarchy as path to the command can allow for specific targets to be identified without having to state it on the command line.
- Has auto-complete for commands, similar to Unix/Linux autocomplete and provides support for command option help as well.
- **Callback functions for commands are simply just argc/argv like functions.**
  - The CLI does not convert arguments for the user, it is up to the developer to decode the argv[] values.
  - Most of the arguments converted in the current cmdline are difficult to use or not required as the developer just picks string type and does the conversion himself.
- Dynamically be able to add and remove commands, directories, files and aliases, does not need to be statically compiled into the application.
- No weird structures in the code and reduces the line count for testpmd from 12K to 4.5K lines. I convert testpmd to have both CMDLINE and CLI with a command line option.
- **Two methods to parse command lines, first is the standard argc/argv method in the function.**
  - The second method is to use a map of strings with simple printf like formatting to detect which command line the user typed.
  - An ID value it returned to the used to indicate which mapping string was found to make the command line to be used in a switch statement.
- Environment variable support using the **env** command or using an API.
- Central help support if needed (optional).

9.1 Overview

The CLI library is a simple set of APIs which allow the developer to quickly create a set of commands using a simple programming interface already familar to the developer.

One of the big advantages of CLI over Cmdline is it is dynamic, which means nodes or items can be added and removed on the fly. Which allows adding new directories, file or commands as needed or removing these items at runtime. The CLI has no global modifiable variables except for the one global pointer which is a thread based variable. Allowing the developer to have multiple CLI instances running at the same time on different threads if needed.

Another big advantage is the calling of the backend function to support a command is very familar to developers as it is basically just a argc/argv style command and the developer gets the complete command line. The function as access to the global thread variable called **this_cli** pointing to the struct cli variable.

9.2 Mapping commands

One other big advantage is the use of MAP structures, to help identify commands quickly plus allowing the developer to define new versions of commands and be able to identify these new versions using a simple identifier value.

The format of the struct cli_map is:

```
struct cli_map show_map[] = {
    /* Index value, Mapping string */
    { 10, "show" },
    { 20, "show %s" },
    { 30, "show %P stats" },
    { 40, "show %P %|link|errors|missed stats" },
    { 0, NULL}
}
```

The map is just an array of struct cli_map entries with a unique index value and mapping string. The index value can be any value the developer wants. As the index value is used to identify the given map string.

The map string is a special formatted string similar to sprintf(), but the format values for % is different. Please look at the cli_mapping() function docs for more information. The %s is for any string and %P is used to a portlist format e.g. 1-3,5-7,9 as used for DPDK command line notation.

The above array is parsed to match the command line from the user. The first map string that matches the user input will be returned from the call to cli_mapping() function.

Constant values are required in the command as in index 30 'stats'. The index 40 is using a variable fixed set of strings option, which means one of these fixed strings must match in that position.

Another advantage of CLI is how simple it is to add new directroies, files and commands for user development. To add a command a developer needs to add an entry to the cli_tree structure and create a function using the above prototype format.

```
struct cli_tree my_tree[] = {
    c_dir("/bin"),
    c_cmd("hello", hello_cmd, "simple hello world command"),
    c_cmd("show",  show_cmd,  "Show system information"),
    c_end()
};
```

The cli_tree structure is made with unions and the c_dir(), c_cmd() and c_end() help initialize the structure easily for the developer. The help and show commands above use the simple argc/argv prototype above.

Only two things are required to create a command a cli_tree entry and a function to call. Using the cli_map and other structures are optional to make adding simple commands quick and easy. The call the cli_create() command or one of its helper functions cli_create_XYZ(). If have a function per command then using the mapping structure is optional, unless you want to have CLI parse and map commands to the exact entries. If cli_map is not used then the developer needs to decode the argc/argv to determine the command requests.

The argc/argv is exactly like the standard usage in a Unix* system, which allows for using getopt() and other standard functions. The Cmdline structures and text conversions were defined at compile time in most cases, but in CLI the command routine is passed the argc/argv

information to convert the strings as needed. The cli variable being a thread Local Storage (TLS) all user routines can access **this_cli** to gain access to the CLI structure if required at all.

9.3 Environment variables

The user can also set environment variables using the **env** command. These variables are also parsed in the command line as direct substitutions.

Another special file is a string file, which can be used as an environment variable. When the variable is asked for the variable asks a function to return the string. The value of the string normally a system value or a generated value. These types of environment variables can not be set from the command line as a function pointer needs to be given. The c_str() macro helps in setting up these environment variables via the cli_tree structure.

The special file backed environment variable can be deleted, but can not be restored without a reboot or some other command puting that variable back into the environment.

Environment variables are denoted by a $(foo) like syntax and are expanded at the time of execution each time the command line is executed. Which means history lines with environment variables will be expanded again.

9.4 Simple Files

The CLI system also has support for simple files along with alias like commands. These simple files are backed by a function call and the other commands can read these files to get constant data or generated data depending on how the backend function works.

9.5 Alias commands

The alias commands are fixed strings which are executed instead of a function provided by the developer. If the user has more arguments these are appended to the alias string and processed as if typed on the command line. Also the environment variables are expanded at execution time.

Note: The CLI library was designed to be used in production code and the Cmdline was not validated to the same standard as other DPDK libraries. The goal is to provide a production CLI design.

The CLI library supports some of the features of the Cmdline library such as, completion, cut/paste and some other special bindings that make configuration and debug faster and easier.

The CLI desin uses some very simple VT100 control strings for displaying data and accepting input from the user. Some of the control strings are used to clear the screen or line and position the cursor on a VT100 compatible terminal. The CLI screen code also supports basic color and many other VT100 commands.

The example application also shows how the CLI application can be extended to handle a list of commands and user input.

The example presents a simple command prompt **DPDK-cli:/>** similar to a Unix* shell command along with a directory like file system.

Some of the **default** commands contained under /sbin directory are:

- **ls**: list the current or provided directory files/commands.
- **cd**: Change directory command.
- **pwd**: print out the current working directory.
- **history**: List the current command line history if enabled.
- **more**: A simple command to page contents of files.
- **help**: display a the help screen.
- **quit**: exit the CLI application, also **Ctrl-x** will exit as well.
- **mkdir**: add a directory to the current directory.
- **delay**: wait for a given number of microseconds.
- **sleep**: wait for a given number of seconds.
- **rm**: remove a directory, file or command. Removing a file will delete the data.
- **cls**: clear the screen and redisplay the prompt.
- **version**: Display the current DPDK version being used.
- **path**: display the current search path for executable commands.
- **cmap**: Display the current system core and socket information.
- **hugepages**: Display the current hugepage information.
- **sizes**: a collection system structure and buffer sizes for debugging.
- **copyright**: a file containing DPDK copyright information.
- **env**: a command show/set/modify the environment variables.
- **ll**: an alias command to display long ls listing **ls -l**
- **h**: alias command for **history**
- **hello**: a simple Hello World! command.
- **show**: has a number of commands using the map feature.

Under the /data directory is:

- **pci**: a simple example file for displaying the **lspci** command in CLI.

Note: To terminate the application, use **Ctrl-x** or the command **quit**.

9.6 Auto completion

CLI does support auto completion at the file or directory level, meaning the arguments to commands are not expanded as was done in Cmdline code. The CLI auto completion works

similar to the standard Unix* system by expanding commands and directory paths. In normal Unix* like commands the user needs to execute the command asking for help information.

9.7 Special command features

Using the '!' followed by a number from the history list of commands you can execute that command again. Or using the UP/Down arrows the user can quickly find and execute or modify a previous command in history.

The user can also execute host level commands if enabled using the '@' prefix to a command line e.g. @ls or @lspci or ... line is passed to popen or system function to be executed and the output displayed on the console if any output.

9.8 Compiling the Application

1. Go to example directory:

```
export RTE_SDK=/path/to/rte_sdk
cd ${RTE_SDK}/examples/cli
```

1. Set the target (a default target is used if not specified). For example:

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

Refer to the *DPDK Getting Started Guide* for possible RTE_TARGET values.

1. Build the application:

```
make
```

9.9 Running the Application

To run the application in linux environment, issue the following command:

```
$ ./build/cli
```

Note: The example cli application does not require to be run as superuser as it does not startup DPDK by calling rte_eal_init() routine. Which means it also does not use DPDK features except for a few routines not requiring EAL initialization.

Refer to the *DPDK Getting Started Guide* for general information on running applications and the Environment Abstraction Layer (EAL) options.

9.10 Explanation

The following sections provide some explanation of the code.

9.10.1 EAL Initialization and cmdline Start

The first task is the initialization of the Environment Abstraction Layer (EAL), if required for the application.

```
int
main(int argc, char **argv)
{
    if (cli_create_with_tree(init_tree) ==0) {
        cli_start(NULL, 0); /* NULL is some init message done only once */
                            /* 0 means do not use color themes */
        cli_destroy();
    }
```

The cli_start() function returns when the user types **Ctrl-x** or uses the quit command in this case, the application exits. The cli_create() call takes four arguments and each has a default value if not provided. The API used here is the cli_create_with_tree(), which uses defaults for three of the arguments.

```
/**
* Create the CLI engine
*
* @param prompt_func
*   Function pointer to call for displaying the prompt.
* @param tree_func
*   The user supplied function to init the tree or can be NULL. If NULL then
*   a default tree is initialized with basic commands.
* @param nb_entries
*   Total number of commands, files, aliases and directories. If 0 then use
*   the default number of nodes. If -1 then unlimited number of nodes.
* @param nb_hist
*   The number of lines to keep in history. If zero then turn off history.
*   If the value is CLI_DEFAULT_HISTORY use CLI_DEFAULT_HIST_LINES
* @return
*   0 on success or -1
*/
int cli_create(cli_prompt_t prompt_func, cli_tree_t tree_func,
                    int nb_entries, uint32_t nb_hist);
```

The cli_create_with_tree() has only one argument which is the structure to use in order to setup the initial directory structure. Also the wrapper function int cli_create_with_defaults(void) can be used as well.

Consult the cli.h header file for the default values. Also the alias node is a special alias file to allow for aliasing a command to another command.

The tree init routine is defined like:

```
static struct cli_tree my_tree[] = {
    c_dir("/data"),
    c_file("pci", pci_file, "display lspci information"),
    c_dir("/bin"),
    c_cmd("hello", hello_cmd, "Hello-World!!"),
    c_alias("h", "history", "display history commands"),
    c_alias("ll", "ls -l", "long directory listing alias"),
    c_end()
};

static int
init_tree(void)
```

(continues on next page)

(continued from previous page)

```
{
    /*
     * Root is created already and using system default cmds and dirs, the
     * developer is not required to use the system default cmds/dirs.
     */
    if (cli_default_tree_init())
        return -1;

        /* Using NULL here to start at root directory */
    if (cli_add_tree(NULL, my_tree))
        return -1;

        cli_help_add("Show", show_map, show_help);

        return cli_add_bin_path("/bin");
}
```

The above structure is used to create the tree structure at initialization time. The struct cli_tree or cli_tree_t typedef can be used to setup a new directory tree or agument the default tree.

The elements are using a set of macros c_dir, c_file, c_cmd, c_alias and c_end. These macros help fill out the cli_tree_t structure for the given type of item.

The developer can create his own tree structure with any commands that are needed and/or call the cli_default_tree_init() routine to get the default structure of commands. If the developer does not wish to call the default CLI routine, then he must call the cli_create_root() function first before adding other nodes. Other nodes can be added and removed at anytime.

9.10.2 CLI Map command support

The CLI command has two types of support to handle arguments normal argc/argv and the map system. As shown above the developer creates a directory tree and attaches a function to a command. The function takes the argc/argv as arguments and the developer can just parse the arguments to decode the command arguments. Sometimes you have multiple commands or different versions of a command being handled by a single routine, this is were the map support comes into play.

The map support defines a set of struct cli_map map[]; to help detect the correct command from the user. In the list of cli_map structures a single structure contains two items a developer defined index value and a command strings. The index value is used on the function to identify the specific type of command found in the list. The string is a special printf like string to help identify the command typed by the user. One of the first things todo in the command routine is to call the cli_mapping() function passing in the CLI pointer and the argc/argv values.The two method can be used at the same time.

The cli_mapping() command matches up the special format string with the values in the argc/argv array and returns the developer supplied index value or really the pointer the struct cli_map instance.

Now the developer can use the cli_map.index value in a switch() statement to locate the command the user typed or if not found a return of -1.

Example:

```
static int
hello_cmd(int argc, char **argv)
{
    int i, opt;

    optind = 1;
    while((opt = getopt(argc, argv, "?")) != -1) {
        switch(opt) {
            case '?': cli_usage(); return 0;
            default:
                break;
        }
    }

    cli_printf("Hello command said: Hello World!! ");
    for(i = 1; i < argc; i++)
        cli_printf("%s ", argv[i]);
    cli_printf("\n");

    return 0;
}

static int
pci_file(struct cli_node *node, char *buff, int len, uint32_t opt)
{
        if (is_file_open(opt)) {
                FILE *f;

                if (node->file_data && (node->fflags & CLI_FREE_DATA))
                        free(node->file_data);

        node->file_data = malloc(32 * 1024);
                if (!node->file_data)
                        return -1;
        node->file_size = 32 * 1024;
        node->fflags = CLI_DATA_RDONLY | CLI_FREE_DATA;

                f = popen("lspci", "r");
                if (!f)
                        return -1;

                node->file_size = fread(node->file_data, 1, node->file_size, f);

                pclose(f);
        return 0;
    }
    return cli_file_handler(node, buff, len, opt);
}

static struct cli_map show_map[] = {
        { 10, "show %P" },
        { 20, "show %P mac %m" },
        { 30, "show %P vlan %d mac %m" },
        { 40, "show %P %|vlan|mac" },
        { -1, NULL }
};

static const char *show_help[] = {
        "show <portlist>",
        "show <portlist> mac <pg_ether_addr>",
        "show <portlist> vlan <vlanid> mac <pg_ether_addr>",
        "show <portlist> [vlan|mac]",
```

(continues on next page)

(continued from previous page)

```
        CLI_HELP_PAUSE,
        NULL
};

static int
show_cmd(int argc, char **argv)
{
        struct cli_map *m;
        uint32_t portlist;
        struct pg_ether_addr mac;

        m = cli_mapping(Show_info.map, argc, argv);
        if (!m)
                return -1;

        switch(m->index) {
                case 10:
                        portlist_parse(argv[1], &portlist);
                        cli_printf("   Show Portlist: %08x\n", portlist);
                        break;
                case 20:
                        portlist_parse(argv[1], &portlist);
                        pg_ether_aton(argv[3], &mac);
                        cli_printf("   Show Portlist: %08x, MAC: %02x:%02x:%02x:
↪%02x:%02x:%02x\n",
                                                portlist,
                                                mac.addr_bytes[0],
                                                mac.addr_bytes[1],
                                                mac.addr_bytes[2],
                                                mac.addr_bytes[3],
                                                mac.addr_bytes[4],
                                                mac.addr_bytes[5]);
                        break;
                case 30:
                        portlist_parse(argv[1], &portlist);
                        pg_ether_aton(argv[5], &mac);
                        cli_printf("   Show Portlist: %08x vlan %d MAC: %02x:%02x:
↪%02x:%02x:%02x:%02x\n",
                                                portlist,
                                                atoi(argv[3]),
                                                mac.addr_bytes[0],
                                                mac.addr_bytes[1],
                                                mac.addr_bytes[2],
                                                mac.addr_bytes[3],
                                                mac.addr_bytes[4],
                                                mac.addr_bytes[5]);
                        break;
                case 40:
                        portlist_parse(argv[1], &portlist);
                        pg_ether_aton("1234:4567:8901", &mac);
                        cli_printf("   Show Portlist: %08x %s: ",
                                                portlist, argv[2]);
                        if (argv[2][0] == 'm')
                                cli_printf("%02x:%02x:%02x:%02x:%02x:%02x\n",
                                                mac.addr_bytes[0],
                                                mac.addr_bytes[1],
                                                mac.addr_bytes[2],
                                                mac.addr_bytes[3],
                                                mac.addr_bytes[4],
                                                mac.addr_bytes[5]);
                        else
```

(continues on next page)

(continued from previous page)

```
                                cli_printf("%d\n", 101);
                        break;
                default:
                        cli_help_show_group("Show");
                        return -1;
        }
        return 0;
}

static struct cli_tree my_tree[] = {
        c_dir("/data"),
    c_file("pci",       pci_file,       "display lspci information"),
    c_dir("/bin"),
    c_cmd("show",       show_cmd,       "show mapping options"),
    c_cmd("hello",      hello_cmd,      "Hello-World!!"),
    c_alias("h",        "history",      "display history commands"),
    c_alias("ll",       "ls -l",        "long directory listing alias"),
    c_end()
};
```

Here is the cli_tree for this example, note it has a lot more commands. The show_cmd or **show** command is located a number of lines down. The cli_tree creates in the /**bin** directory a number of commands and the show command is one of these. The show command has four different formats if you look at the **show_map[]** structure.

The user types one of these commands and cli_mapping() function attempts to locate the correct entry in the list. You will also notice another structure called **show_help**, which is an array of strings giving a cleaner and longer help description of each of the commands.

9.10.3 Understanding the CLI system

The command line interface is defined as a fake directory tree with executables, directorys and files. The user uses shell like standard commands to move about the directory and execute commands. The CLI is not a powerful as the Bash shell, but has a number of similar concepts.

Our fake directory tree has a '/' or root directory which is created when cli_create() is called along with the default sbin directory. The user starts out at the root directory '/' and is allowed to cd to other directories, which could contain more executables, aliases or directories. The max number of directory levels is limited to the number of nodes given at startup.

The default directory tree starts out as just root (/) and a sbin directory. Also it contains a file called copyright in root, which can be displayed using the default 'more copyright' command.

A number of default commands are predefined in the /sbin directory and are defined above. Other bin directories can be added to the system if needed, but a limit of CLI_MAX_BINS is defined in the cli.h header file.

CHAPTER 10

Running Script Files

Pktgen can read and run files with default values and configurations via the `-f` commandline option (*Pktgen Commandline Options*).

These files can either be `.pkt` files with Pktgen *runtime commands* as shown in the previous section or `.lua` files with the same commands and options in Lua syntax.

For example here is a `pktgen` instance that read a `.pkt` file:

```
pktgen -l 0-4 -n 3 --proc-type auto --socket-mem 128,128 -- \
             -P -m "[1:3].0, [2:4].1" -f test/set_seq.pkt
```

Where the `test/set_seq.pkt` (included in the `pktgen` repository) is as follows:

```
seq 0 all 0000:4455:6677 0000:1234:5678 10.11.0.1 10.10.0.1/16 5 6 ipv4 udp 1 128
set all seqCnt 1
```

The Lua version (`test/set_seq.lua` in `pktgen` repository) is clearer and allows extension through standard Lua or user defined functions:

```
-- manual for more comment options.
require "Pktgen"
local seq_table = {         -- entries can be in any order
    ["eth_dst_addr"] = "0011:4455:6677",
    ["eth_src_addr"] = "0011:1234:5678",
    ["ip_dst_addr"] = "10.12.0.1",
    ["ip_src_addr"] = "10.12.0.1/16",   -- the 16 is the size of the mask value
    ["sport"] = 9,          -- Standard port numbers
    ["dport"] = 10,         -- Standard port numbers
    ["ethType"] = "ipv4",   -- ipv4|ipv6|vlan
    ["ipProto"] = "udp",    -- udp|tcp|icmp
    ["vlanid"] = 1,         -- 1 - 4095
    ["pktSize"] = 128,      -- 64 - 1518
    ["teid"] = 3,
    ["cos"] = 5,
    ["tos"] = 6
  };
-- seqTable( seq#, portlist, table );
```

(continues on next page)

(continued from previous page)

```
pktgen.seqTable(0, "all", seq_table );
pktgen.set("all", "seq_cnt", 1);
```

The Lua interface is explained in the next section *Using Lua with Pktgen*.

CHAPTER 11

Using Lua with Pktgen

Lua is a high level dynamic programming language. It is small and lightweight and can easily be embedded in applications written in other languages. It is also suitable for loading and wrapping dynamic libraries.

Lua is used in `pktgen` to script and configure the application and also to plug into DPDK functions to expose configuration and statistics.

The following are some of the examples included in the `test` directory of `pktgen` repository.

11.1 Example: Hello World

A simple "hello world" example to ensure that everything is working correctly:

```
package.path = package.path ..";?.lua;test/?.lua;app/?.lua;"

require "Pktgen"
printf("Lua Version      : %s\n", pktgen.info.Lua_Version);
printf("Pktgen Version   : %s\n", pktgen.info.Pktgen_Version);
printf("Pktgen Copyright : %s\n", pktgen.info.Pktgen_Copyright);
printf("Pktgen Authors   : %s\n", pktgen.info.Pktgen_Authors);

printf("\nHello World!!!!\n");
```

11.2 Example: Info

A simple example to print out some metadata and configuration information from `pktgen`:

```
package.path = package.path ..";?.lua;test/?.lua;app/?.lua;"

require "Pktgen"

-- A list of the test script for Pktgen and Lua.
```

(continues on next page)

(continued from previous page)

```
-- Each command somewhat mirrors the pktgen command line versions.
-- A couple of the arguments have be changed to be more like the others.
--
function info()
        printf("Lua Version      : %s\n", pktgen.info.Lua_Version);

        printf("Pktgen Version   : %s\n",
                pktgen.info.Pktgen_Version);
        printf("Pktgen Copyright : %s\n",
                pktgen.info.Pktgen_Copyright);

        prints("pktgen.info",
                pktgen.info);

        printf("Port Count %d\n",
                pktgen.portCount());
        printf("Total port Count %d\n",
                pktgen.totalPorts());
end

info()
```

11.3 Example: More Info

Another example to print out data from a running `pktgen` instance:

```
package.path = package.path ..";?.lua;test/?.lua;app/?.lua;"

require "Pktgen"
-- A list of the test script for Pktgen and Lua.
-- Each command somewhat mirrors the pktgen command line versions.
-- A couple of the arguments have be changed to be more like the others.
--

prints("linkState", pktgen.linkState("all"));
prints("isSending", pktgen.isSending("all"));
prints("portSizes", pktgen.portSizes("all"));
prints("pktStats", pktgen.pktStats("all"));
prints("portRates", pktgen.portStats("all", "rate"));
prints("portStats", pktgen.portStats("all", "port"));
```

11.4 Example: Sequence

An example to set a packet sequence:

```
package.path = package.path ..";?.lua;test/?.lua;app/?.lua;"
-- Lua uses '--' as comment to end of line read the
-- manual for more comment options.
require "Pktgen"
local seq_table = {         -- entries can be in any order
    ["eth_dst_addr"] = "0011:4455:6677",
    ["eth_src_addr"] = "0011:1234:5678",
    ["ip_dst_addr"] = "10.12.0.1",
    ["ip_src_addr"] = "10.12.0.1/16",   -- the 16 is the size of the mask value
```

(continues on next page)

(continued from previous page)

```
    ["sport"] = 9,          -- Standard port numbers
    ["dport"] = 10,         -- Standard port numbers
    ["ethType"] = "ipv4",   -- ipv4|ipv6|vlan
    ["ipProto"] = "udp",    -- udp|tcp|icmp
    ["vlanid"] = 1,         -- 1 - 4095
    ["pktSize"] = 128,      -- 64 - 1518
    ["teid"] = 3,
    ["cos"] = 5,
    ["tos"] = 6
  };
-- seqTable( seq#, portlist, table );
pktgen.seqTable(0, "all", seq_table );
pktgen.set("all", "seq_cnt", 1);
```

11.5 Example: Main

A more complex example showing most of the features available via the Lua interface and also show interaction with the user:

```
package.path = package.path ..";?.lua;test/?.lua;app/?.lua;"

require "Pktgen"

-- A list of the test script for Pktgen and Lua.
-- Each command somewhat mirrors the pktgen command line versions.
-- A couple of the arguments have be changed to be more like the others.
--
pktgen.screen("off");
pktgen.pause("Screen off\n", 1000);
pktgen.screen("on");
pktgen.pause("Screen on\n", 1000);
pktgen.screen("off");
pktgen.pause("Screen off\n", 1000);

printf("delay for 1 second\n");
pktgen.delay(1000);
printf("done\n");

-- 'set' commands for a number of per port values
pktgen.set("all", "count", 100);
pktgen.set("all", "rate", 50);
pktgen.set("all", "size", 256);
pktgen.set("all", "burst", 128);
pktgen.set("all", "sport", 0x5678);
pktgen.set("all", "dport", 0x9988);
pktgen.set("all", "prime", 3);
pktgen.set("all", "seq_cnt", 3);

pktgen.rnd("all", 1, 20, "XX111000.. ..xx11");

pktgen.vlanid("all", 55);

pktgen.screen("on");
pktgen.pause("Screen on\n", 1000);
pktgen.screen("off");

-- sequence command in one line
```

(continues on next page)

(continued from previous page)

```
pktgen.seq(0, "all", "0000:4455:6677", "0000:1234:5678", "10.11.0.1", "10.10.0.1/16
↪", 5, 6, "ipv4", "udp", 1, 128);
prints("seq", pktgen.decompile(0, "all"));

-- sequence command using a table of packet configurations
local seq_table = {
    ["eth_dst_addr"] = "0011:4455:6677",
    ["eth_src_addr"] = "0011:1234:5678",
    ["ip_dst_addr"] = "10.12.0.1",
    ["ip_src_addr"] = "10.12.0.1/16",
    ["sport"] = 9,
    ["dport"] = 10,
    ["ethType"] = "ipv4",
    ["ipProto"] = "udp",
    ["vlanid"] = 1,
    ["pktSize"] = 128
  };
pktgen.seqTable(0, "all", seq_table );

prints("seqTable", pktgen.decompile(0, "all"));

pktgen.ports_per_page(2);
pktgen.icmp_echo("all", "on");
pktgen.send_arp("all", "g");
pktgen.set_mac("0-2","src", "0001:1122:3344");
pktgen.set_mac("0-2", "dst","0002:1122:3344");
pktgen.mac_from_arp("on");
pktgen.set_ipaddr("0", "dst", "10.10.2.2");
pktgen.set_ipaddr("0", "src", "10.10.1.2/24");
pktgen.set_ipaddr("1", "dst", "10.10.2.2");
pktgen.set_ipaddr("1", "src", "10.10.2.2/24");
pktgen.set_proto("all", "udp");
pktgen.set_type("all", "ipv6");
pktgen.ping4("all");
--pktgen.ping6("all");
--pktgen.show("all", "scan");
pktgen.pcap("all", "on");
pktgen.ports_per_page(4);
pktgen.start("all");
pktgen.stop("all");
pktgen.prime("all");
pktgen.delay(1000);

pktgen.screen("on");
pktgen.clear("all");
pktgen.cls();
pktgen.reset("all");

pktgen.pause("Do range commands\n", 1000);
pktgen.page("range");
pktgen.dst_mac("all", "start", "0011:2233:4455");
pktgen.src_mac("all", "start", "0033:2233:4455");

pktgen.delay(1000);
pktgen.dst_ip("all", "start", "10.12.0.1");
pktgen.dst_ip("all", "inc", "0.0.0.2");
pktgen.dst_ip("all", "min", "10.12.0.1");
pktgen.dst_ip("all", "max", "10.12.0.64");

pktgen.delay(1000);
pktgen.src_ip("all", "start", "10.13.0.1");
```

(continues on next page)

(continued from previous page)

```
pktgen.src_ip("all", "inc", "0.0.0.3");
pktgen.src_ip("all", "min", "10.13.0.1");
pktgen.src_ip("all", "max", "10.13.0.64");

pktgen.delay(1000);
pktgen.dst_port("all", "start", 1234);
pktgen.dst_port("all", "inc", 4);
pktgen.dst_port("all", "min", 1234);
pktgen.dst_port("all", "max", 2345);

pktgen.delay(1000);
pktgen.src_port("all", "start", 5678);
pktgen.src_port("all", "inc", 5);
pktgen.src_port("all", "min", 1234);
pktgen.src_port("all", "max", 9999);

pktgen.delay(1000);
pktgen.vlan_id("all", "start", 1);
pktgen.vlan_id("all", "inc", 0);
pktgen.vlan_id("all", "min", 1);
pktgen.vlan_id("all", "max", 4094);

pktgen.delay(1000);
pktgen.pkt_size("all", "start", 128);
pktgen.pkt_size("all", "inc", 2);
pktgen.pkt_size("all", "min", 64);
pktgen.pkt_size("all", "max", 1518);

pktgen.pause("Wait a second, then go back to main page\n", 1000);

pktgen.page("0");
pktgen.pause("About to do range\n", 1000);
pktgen.set_range("all", "on");

pktgen.port(2);
pktgen.process("all", "on");
pktgen.blink("0", "on");
pktgen.pause("Pause for a while, then turn off screen\n", 4000);
pktgen.screen("off");

printf("Lua Version      : %s\n", pktgen.info.Lua_Version);
printf("Pktgen Version   : %s\n", pktgen.info.Pktgen_Version);
printf("Pktgen Copyright : %s\n", pktgen.info.Pktgen_Copyright);

prints("pktgen.info", pktgen.info);

printf("Port Count %d\n", pktgen.portCount());
printf("Total port Count %d\n", pktgen.totalPorts());

printf("\nDone\n");
key = pktgen.continue("\nPress any key: ");
if ( key == "s" ) then
        pktgen.set("all", "seq_cnt", 4);
        pktgen.save("foobar.cmd");
        pktgen.continue("Saved foobar.cmd, press key to load that file: ");
        pktgen.load("foobar.cmd");
end
```

CHAPTER 12

Socket Support for Pktgen

Pktgen provides a TCP socket connection to allow you to control it from a remote console or program.

The TCP connection uses port 22022, 0x5606, and presents a Lua command shell interface.

If you telnet on port 22022 to a machine running `pktgen` you will get a Lua command shell like interface. This interface does not have a command line prompt, but you can issue Lua code or load script files from the local disk of the machine. You can also send programs to the remote `pktgen` machine to load scripts and run scripts.

Another way to connect remotely to `pktgen` is to use the `socat` program on a Linux machine:

```
$ socat -d -d READLINE TCP4:localhost:22022
```

This will create a connection and then wait for Lua command scripts. You can also send `pktgen` a command script file and display the output:

```
$ socat - TCP4:localhost:22022 < test/hello-world.lua

Lua Version      : Lua 5.3
Pktgen Version   : 2.9.0
Pktgen Copyright : Copyright (c) <2010-2015>, Wind River Systems, Inc.
Pktgen Authors   : Keith Wiles @ Wind River Systems

Hello World!!!!
```

Where the the `test/hello-world.lua` looks like this:

```
package.path = package.path ..";?.lua;test/?.lua;app/?.lua;"

require "Pktgen"
printf("Lua Version      : %s\n", pktgen.info.Lua_Version);
printf("Pktgen Version   : %s\n", pktgen.info.Pktgen_Version);
printf("Pktgen Copyright : %s\n", pktgen.info.Pktgen_Copyright);
printf("Pktgen Authors   : %s\n", pktgen.info.Pktgen_Authors);

printf("\nHello World!!!!\n");
```

Here is another `socat` example which loads a file from the local disk where `pktgen` is running and then we execute the file with a user defined function:

```
$ socat READLINE TCP4:172.25.40.163:22022
f,e = loadfile("test/hello-world.lua")
f()
Lua Version      : Lua 5.3
Pktgen Version   : 2.9.0
Pktgen Copyright : Copyright (c) <2010-2015>, Wind River Systems, Inc.
Pktgen Authors   : Keith Wiles @ Wind River Systems

Hello World!!!!
<Control-D>
```

You can also just send it commands via echo:

```
$ echo "f,e = loadfile('test/hello-world.lua'); f();" \
       | socat - TCP4:172.25.40.163:22022
Lua Version      : Lua 5.3
Pktgen Version   : 2.9.0
Pktgen Copyright : Copyright (c) <2010-2015>, Wind River Systems, Inc.
Pktgen Authors   : Keith Wiles @ Wind River Systems

Hello World!!!!
```

CHAPTER 13

Changes in Pktgen

This section shows changes and bug fixes in the Pktgen application.

CHAPTER 14

Pktgen-DPDK - Traffic Generator powered by DPDK

Pktgen-DPDK is a traffic generator powered by DPDK at wire rate traffic with 64 byte frames.

** (Pktgen) Sounds like 'Packet-Gen'**

=== Modifications ===

- **19.08.0 - Fix linking of Lua library when no pkg-config file is found and** linking liblua.a Fixed and issue with packet rate not being changed when the packet size is changed Change version numering to year.month.patch format Change lua pktgen.set_mac() to have three args set_mac(<portlist>, 'src|dst', mac_addr) instead of two args add support for setting via lua src and dst mac addresses fix up the RSS port configuration options updated code to adjust the packet tx interval based on command changes on the packet size and other areas.
- **3.7.1 - Add TTL support to single and range modes. Better docs for dump packets**
  - Pktgen 3.7.1 will build with DPDK 18.02 to 19.08-rc2, but 18.08 has a problem with vhost.h VRING_EVENT_F_AVAIL not defined and appears the Ubuntu 19.04 version I am running does not define it correctly.
  - fixed the RX side using TX count for rx_burst command.
  - General cleanup
- **3.7.0 - Fixed build issues with DPDK 19.08 as DPDK renamed a lot of defines** Fixed up the meson files to build pktgen with meson and ninja Minor cleanup
- **3.6.6 - Add portInfo() function in lua to return most of the port info in** one structure.
- 3.6.5 - Fix ldflags order for RHEL, CentOS and other systems to include lua in build
- **3.6.4 - Add a new page stats screen to dump out info** Updated the readme to give better info on .cfg files Fixed the port <N> to include only valid ports fix

port command functions to use uint16_t for ports. fix detection of Lua libs testing LUA_PKT_LIBS for Ubuntu 18.10

- **3.6.3 - Fix race condition in start command** Add support of having a different PCAP file per queue of a port Replace constant with macro for PCAP buffer size Expose number missed frames to Lua stats. Fix cleanup of unsent pkts, this fixes duplicate packets being sent.
- 3.6.2 - Fix GRE header pointer being NULL and fixed IPPROTO_UDP to use PG_IPPROTO_UDP
- 3.6.1 - Fix string truncation warnings for latest GCC
- **3.6.0 - Fixed config files.**
  - Fixed tx stopping on some platforms.
  - Updated the copyrights to 2019
  - Add verbose flag to cleanup startup text.
  - Adding support for VxLAN
- **3.5.9 - Fixed run.py to find config files better.**
  - Limit the pcap packets to the number of mbufs used.
  - Added new 'page xstats' screen to view extended stats for ports.
  - other minor patches
- 3.5.8 - Add code to check for using master lcore for ports
- 3.5.7 - Fix CentOS building with Lua, fixed cli building as was not linking cli
- 3.5.6 - Minor fixes to cleanup builds sync cli,vec,utils and lua code bases.
- 3.5.5 - Minor fixes to run pktgen without libs in DPDK.
- 3.5.4 - Fix possible mbuf leak and clean up colors for pages.
- **3.5.3 - Fix performance problem on FVL NIC with losing mbufs.** Number of little cleanups and fixes.
- **3.5.2 - Fix 18.08-rc0 fixed some fields that were removed txq_flags and .ignore_offload_field** Tested backward compat to 18.02 at least. Pktgen should work with at least 18.02 release.
- 3.5.1 - Fix 18.05 related problems and minor cleanup in port configuration.
- **3.5.0 - fix compiler warning and range size increment command.** fixes for working with dpdk 18.04 release.
- **3.4.9 - fix PRNG random number code to work correctly.** Add more validation of -m commands Fix the rte_pci_bus.h header include problems.
- 3.4.8 - Increase the TX retry count for VMs and reset mbuf as virtio modifies the data.
- 3.4.7 - Command file can now use 'quit' to exit and Lua scripts use pktgen.quit()
- 3.4.6 - Fixed low performance in a VM with virtio and a TX hang.
- 3.4.5 - Fixed a few backward compact issues and synced CLI code

- 3.4.4 - Fixed the long line problem in CLI and improved the performance.
- 3.4.3 - Fix compile problem with DPKD 17.11 and a number of bugs.
- 3.4.2 - Fix a build problem with older DPDK versions.
- 3.4.1 - Fix ARP packet processing and a fe minor cfg file changes.
- 3.4.0 - Fix bonding polling and other minor changes.
- **3.3.9 - PCI Whitelist and Blacklist should be the original defines and** does not require a ifdef for the version. Convert the run.py to dpdk-run.py script and convert the scripts to /bin/sh.
- 3.3.8 - reverse change causing TX to stop sending traffic.
- 3.3.7 - Fix BLACKLIST and WHITELIST macro change for 17.08 DPDK
- **3.3.6 - Add flag to enable bonding PMD to do TX calls with zero packets.** New command 'enable|disable bonding' Cleanup some of the copyright dates to use 2018
- 3.3.5 - fix sequeuce command be truncated and fix run.py to fix strings in cfg file.
- 3.3.4 - Minor fixes for help and run.py scipt.
- 3.3.3 - Make sure the mbuf data size is at least 2K in size
- 3.3.2 - Fix set pkt types and fix setting the pkt size for IPv6
- 3.3.1 - Update the help for range and add compact commands for range
- 3.3.0 - Add new run configs and run.py script to setup and run pktgen
- 3.2.12- Rework the src/dst IP address again and fix the reset command
- 3.2.11- Fixed location of libs for per v17.05 releases
- **3.2.10- Fixed the problem with set <portlist> src/dst ip <addr> needing /XX for subnet** mask values. Add a test for using the master lcore for a port and error out.
- **3.2.9 - fixed a number of problems and some code cleanup** Fixed the problem when running a lua script the ports stats were not updated. The problem is a rte_timer_manage() call was remove my mistake.
- 3.2.8 - fixed 'set <portlist> type ipv4|ipv6' used ip4|ip6 instead.
- 3.2.7 - Fix pktgen.seq() lua function Ethertype and protocol were swapped doc is correct.
- **3.2.6 - Fix the pcap page not displaying.** Fix standalone builds to include Lua headers.
- **3.2.5 - Fix setting of the seq_cnt in lua. Now you can use seqCnt or seq_cnt strings** for setting the sequeuce count value.
- 3.2.4 - Fix setting of MAC and IP addresses in single mode.
- **3.2.3 - Allow pktgen to build on DPDK 14.04** fix the new commands to use the correct syntax
- **3.2.2 - fix the clear stats command.** fix range commands to match help text.
- 3.2.1 - Update the readme file.

- **3.2.0 - Add support for decimal point rate values, like 10.1, 20.54, 90.6, . . .** Convert over to use CLI interface instead of cmdline interface. CLI is a directory like command line tool and please read the .rst file in the lib/cli directory Many bug fixes.
- **3.1.2 - Convert spaces to tabs and add pktgen-cfg.[ch] page** Converting the spaces to tabs to allow for editing the code with tabs set to 4 or 8 columns per tab. Changed to allow people who are stuck on a tab=8 columns. We have modern computers tabs=4 columns seems reasonable.
- 3.1.1 - Minor cleanup of top level directory and code.
- **3.1.0 - Rename functions and files.** When files are written change the file modes to 0666 as they are owned by root. More general cleanup of the display refresh. loading command or lua files is faster because of the screen updates fixed.
- **3.0.17- Fixed a formating issue on sequence page for port numbers.** Save of lua code wrong for pktgen.range() should be pktgen.set_range().
- **3.0.16- Add command line option to strip CRC in hardware one RX.** Option is '–crc-strip' which strips the CRC on RX for all ports.
- **3.0.15- Update Lua to 5.3.3 version187** Change lua pktgen.range() to pktgen.set_range() plus added the range commands from pktgen.dst_mac() to pktgen.range.dst_mac(). Still support the old commands for now. Now polls the RX and TX queues to support eth_bond interfaces using mode 4 or LACP.
- 3.0.14- Fix crash in saving configuration and random is not setup.
- 3.0.13- Fix seq only sending the first sequence packet and some cleanup.
- 3.0.12- Map port/queue pair stats to the correct lcore.
- 3.0.11- Fix compile problem with 16.04
- **3.0.10- Added the 'pdump <portlist>' command to hex dump the first packet to be** send on the given ports. Only the single packet mode is supported.
- 3.0.09- Add Fix for PCAP corruption.
- **3.0.08- Add Lua support for rnd and latency commands.** Now if latency is enable on a port then getting the stats will get the latency values via lua table.
- 3.0.07- Fixed crash on exit when using more then one core per port.
- **3.0.06- Fix PCI null pointer when using virtual devices.** Removed the C11 constructs to compile on C99 compilers. Fix the bug of old packets after changes for new run. The problem is DPDK holds on to the mbufs in the TX done queue and those can not be changed. With 16.07 we can find all of the mbufs and changed them to correct format/sizes.
- **3.0.05- New Latency/Jitter page 'page latency'** Need to adjust the packet size to 96 to allow for latency timestamp. type: page latency latency 0 on set 0 size 96 start 0
- **3.0.04- Display reported the wrong rate in Mbits because the counters were not** including the FCS bytes in the value as it was before. Minor cleanup of the code for formating.

- **3.0.03- General clean up of scripts** Add support for share library builds Clean up formating Add PCI info screen GUI 1.0 support
- **3.0.02- Fix up the IPv6 address macros for musl libc support** Fix the missing pthread.h include in lua socket header. Add the rnd lua support APIs Fix the checksum issue with rnd changes.
- 3.0.01- Fixed the Range sequence and VLAN problem.
- 3.0.00- Fixed code to ifdef the imcasts counters that were deprecated.
- 2.9.18- Fix the range command to set IP proto value and be able to save that value.
- 2.9.17- Fix PCAP crash when using multiple tx queues.
- **2.9.16- Fix include problems with cmdline_parse.h file.** missing cmdline_parse_token_hdr_t define and looks like the header was not included in the pktgen-seq.h file for DPDK v2.2.0 and pktgen-2.9.15
- 2.9.15- Update Lua to version 5.3.2
- 2.9.14- Fix compiler error for gcc-4.9 and inet_ntop() routine
- 2.9.13- Add max last seen RX/TX packets to display.
- **2.9.12- Was not able to set IP protocol type for range packets.** New command range.proto <portlist> udp|tcp|icmp Lua command is pktgen.ip_proto("all", "udp")
- 2.9.11- Fix version string for new version style in DPDK.
- 2.9.10- Reformat the code and get ready for the next release of DPDK.
- 2.9.9 - Update the init screen output to not screw up DPDK screen output.
- 2.9.8 - Fixed the crash when using the sequence packets.
- **2.9.7 - Changed all rte_zmalloc to rte_zmalloc_socket calls and change seq_pkt support.** The fix for multiple cores accessing seq_pkts was to allocate memory and copy into a private area. This sometimes caused memory allocation problems, so I removed the allocation and used spinlocks around the code. Most likely slower in some areas but better then allocating memory.
- **2.9.6 - Add support for different pattern types and a user patten string.** New commands are 'pattern <portlist> type'. Types are abc, none, zero or user New command 'user.pattern <portlist> <string>' The string can not contain a space which is a limitation of the rte_cmdline code. Added new Lua command for the above 'pattern(<portlist>, <type>)' and 'userPatten(<portlist>, <string>)'
- 2.9.5 - Fixed sequence packet race condition for multiple senders.
- 2.9.4 - Fixed the ARP sends were not being flushed
- **2.9.3 - Remove change log and comment out the eth stop when done sending.** This will most likely screw up the pcap and others, but stopping the port is not good.
- 2.9.1 - Fix up the sequeue help to remove vlan option with ipv4/ipv6
- 2.9.0 - Update to DPDK 2.0.0 and Lua 5.3.0 with README update.
- 2.8.6 - Fix argument for rte_mempool_create, which caused a crash.

- 2.8.5 - Fix compat problem with latest Pktgen and DPDK 1.8.0
- 2.8.4 - Minor updates for comments.
- **2.8.3 - Updated the Makefiles to use rte.extXYZ.mk files.** Updated the code to build with DPDK 2.0.0-rc1 as some function prototype changed.
- 2.8.2 - Fix bug in pktgen_main_receive routine not using the correct port number.
- 2.8.1 - Add a new docs directory using Sphinx format and update version numbers.
- 2.8.0 - Update to release 1.8.0 of DPDK.
- **2.7.7 - Update Lua to 5.2.3 and fixed setting vlan ID on single ports plus added new Lua fun** New Lua functions are pktgen.portCount() and pktgen.totalPorts() portCount() is the number of port used by Pktgen and totalPorts() is the total number seen by DPDK.
- 2.7.6 - Update code from dpdk.org version of Pktgen, which hopefull fixes the send foreve problem.
- 2.7.5 - Update to latest dpdk.org and move scrn to lib directory with name changes.
- 2.7.4 - Removed old printf_info() calls for printf_status() calls.
- 2.7.3 - Fixed race condition with updating the TX count value with a small count.
- 2.7.1 - Add a command line option '-T' to enable themes and set themes off by default.
- **2.7.0 - Update to DPDK 1.7.0, Note: DPDK 1.7.0 changed how ports are detected and blacklis** which means the port index is now different. You will need to blacklist or whitelist ports with the DPDK '-b' or '–pci-blacklist or –pci-whitelist' options. Pktgen does not blacklist ports anymore. - Moved pktgen to the examples directory plus removed the libwr_* from the lib directory - Pktgen now supports ANSI color terminals only the main screen ATM, but more later. - Best viewed on a black background display, unless you want to change it with the new theme commands. - More supported generator types, checkout the help screens.
- 2.6.8 - Fixed a transmit problem when count is set to one. Plus increase the link down delays.
- 2.6.7 - Add more support for GRE packets, log support and more testing code.
- **2.6.6 - Fix compile problem when not SSE4.2 instructions are not supported. Allowing QEM** systems to build and run. Also added a patch to take into account huge reserved pages.
- 2.6.5 - Added support for logging packet information.
- **2.6.4 - It consists of 3 commits: improvements to the pktgen-random.c unit tests,** the real CentOS compilation fixes and a small update to tap.{c,h} so they are identical to those from zorgnax/libtap on github.
- **2.6.3 - Add a delay when traffic stops to make sure all packets are sent.** Remove the *rte_hash_crc.h* include in wr_pcap.c file.
- 2.6.2 - Fixup GRE and ARP problems
- 2.6.1 - Add random bits support and more cleanup
- 2.6.0 - Split up the code for testing to be added later

- 2.5.2 - Remove extra ethertypes.h file.
- **2.5.1 - Added the following updates.**
  - Implement-Rx-packet-dump-functionality
  - Add-packet-capture-functionality
  - Add-MPLS-functionality
  - Add-Q-in-Q-802.11ad-functionality
  - Add-GRE-header-generation
  - Fix-UDP-TCP-ICMP-protocol-selection
  - Add-ARP-protocol
- 2.5.0 - Update to DPDK 1.6.0 plus a few bug fixes.
- 2.4.1 - Fixed a bug in range packets when 'inc' value is zero use start values.
- 2.4.0 - Add support for TX tap packets. Change 'tap' command to rxtap and txtap.
- 2.3.4 - Minor update to help eliminate RX errors and be able to receive at wire rate.
- 2.3.3 - Update to minor release 1.5.2
- 2.3.2 - Fixed VLAN detection problem in ARP and special GARP support.
- 2.3.1 - Getting closer to line rate tx speed.
- **2.3.0 - Now supports the VLAN encapsulated packets for ARP replies** Also added a special GARP processing to update the destination MAC address to help support a special request for failover support. Added support for DPDK 1.5.1
- 2.2.7 - Updated the code to handle multiple TX queues per port.
- 2.2.6 - Fixed a crash if the port is not up with link status
- 2.2.5 - Remove the flow control code as some systems it does not work.
- 2.2.4 - Fix the *inet_h64tom and inet_mtoh64* functions to account for endianness
- 2.2.3 - range packet fixes for packet size and source/destination mac
- 2.2.2 - Minor performance changes for receive packet performance.
- **2.2.1 - Change MAC address from XXXX:XXXX:XXXX to XX:XX:XX:XX:XX:XX format** Fixed Pktgen to allow packet changes without having to restart the tool.
- 2.2.0 - Update to DPDK 1.5.0

==========

CHAPTER 15

Copyright and License

CHAPTER 16

Third Party License Notices

This document contains third party intellectual property (IP) notices for the Intel Corp® Packet Generation distribution. Certain licenses and license notices may appear in other parts of the product distribution in accordance with the license requirements. “Intel Corp”, is a registered trademark of Intel Corp. All other third-party trademarks are the property of their respective owners.

16.1 Lua

Lua Version 5.3.0.

Lua is a lightweight, embeddable scripting language. Lua combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics. Lua is dynamically typed, runs by interpreting bytecode for a register-based virtual machine, and has automatic memory management with incremental garbage collection, making it ideal for configuration, scripting, and rapid prototyping.

Copyright (c) 1994-2015 Lua.org, PUC-Rio.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and tom permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTH-

ERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

16.2 Warranty Disclaimer & Limitation of Liability

OPEN SOURCE SOFTWARE: “Open Source Software” is software that may be delivered with the Software and is licensed in accordance with open source licenses, including, but not limited to, any software licensed under Academic Free License, Apache Software License, Artistic License, BSD License, GNU General Public License, GNU Library General Public License, GNU Lesser Public License, Mozilla Public License, Python License or any other similar license.

DISCLAIMER OF WARRANTIES: WIND RIVER AND ITS LICENSORS DISCLAIM ALL WARRANTIES, EXPRESS, IMPLIED AND STATUTORY INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF THIRD PARTY RIGHTS WITH RESPECT TO OPEN SOURCE SOFTWARE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY WIND RIVER, ITS DEALERS, DISTRIBUTORS, AGENTS OR EMPLOYEES SHALL IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. Some jurisdictions do not allow the limitation or exclusion of implied warranties or how long an implied warranty may last, so the above limitations may not apply to Customer. This warranty gives Customer specific legal rights and Customer may have other rights that vary from jurisdiction to jurisdiction.

LIMITATION OF LIABILITY: WIND RIVER AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, SPECIAL, CONSEQUENTIAL OR INDIRECT DAMAGES OF ANY KIND (INCLUDING DAMAGES FOR INTERRUPTION OF BUSINESS, PROCUREMENT OF SUBSTITUTE GOODS, LOSS OF PROFITS, OR THE LIKE) REGARDLESS OF THE FORM OF ACTION WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT PRODUCT LIABILITY OR ANY OTHER LEGAL OR EQUITABLE THEORY, ARISING OUT OF OR RELATED TO OPEN SOURCE SOFTWARE, EVEN IF WIND RIVER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT WILL WIND RIVER’S AGGREGATE CUMULATIVE LIABILITY FOR ANY CLAIMS ARISING OUT OF OR RELATED TO OPEN SOURCE SOFTWARE EXCEED ONE HUNDRED DOLLARS ($100). Some jurisdictions do not allow the exclusion or limitation of incidental or consequential damages so this limitation and exclusion may not apply to Customer.

THE WARRANTY DISCLAIMER AND LIMITED LIABILITY ARE FUNDAMENTAL ELEMENTS OF THE BASIS OF THE BARGAIN BETWEEN WIND RIVER AND CUSTOMER. WIND RIVER WOULD NOT BE ABLE TO PROVIDE OPEN SOURCE SOFTWARE WITHOUT SUCH LIMITATIONS.