High Performance Computing and Networking Research Group
Universidad Autónoma de Madrid
Escuela Politécnica Superior

User guide

HPCAP

Víctor Moreno Martínez
victor.moreno@uam.es

Madrid, December 22, 2014

Contents

1 Using the HPCAP driver

The first step you must follow to use HPCAP in your system is obtain the HPCAP`X` (where `X` = release number) containing all the files related to the corresponding release. Inside this folder you will find:

1.1 Installing all the required packages

Inside the `deps` folder you will a `install.bash` script that will install the required packages in a Debian-based distro. It is possible to use both the HPCAP driver and the detect-Pro10G in a different distro, but you will have to manually install the required dependencies.

1.2 Configure your installation

All the scripts used for the installation and management of the HPCAP driver make use of the information specified in the `params.cfg` file that is located in the root level of the HPCAP`X` folder. Consequently, this file has to be properly modified so the HPCAP driver and dependant applications can properly run in your system.

Here you can find a list with parameters you must make sure to have properly configured before you run HPCAP:

- `basedir`: this parameter contains the path to your installation. For example, if you install HPCAP in the `home` folder of user `foo` the value that must be written in this file is: `/home/foo/HPCAPX`.
- `nif`: this parameter must contain the number of network interfaces available in your system (only the ones that would be managed by Intel's `ixgbe` driver). This number is ussually two times the number of PCI network cards plugged in your system (assuming you plug only 2-port cards), but this could vary if you use, for example, 1-port, 4-port network cards.
- `nrxq,ntxq`: number of RSS/Flow-Director queues that you want to use in your 10Gb/s network interfaces for both RX and TX purposes. The default value for this parameter is 1, which is recommended to be kept unless you know what changing this value implies.
- `ifs`: this is the list of interfaces that you want to be automatically woken up once the HPCAP driver has been installed. For each of those interfaces a monitoring script will be launched and will keep record of the received and lost packets and bytes (inside the `data` subfolder, see 1.4). Check 1.3 for more information regarding the interface naming and numbering policy. **Warning:** only a subset those interfaces configured to work in HPCAP mode should appear on this list (no standard-working interfaces).
- Interface-related parameters: those parameters must be configured for each one of the interfaces listed by the `ifs` parameter. All those parameters follow the format `<param_name><itf_index>` where `<itf_index>` is the number identifying the index regardless the prefix to that number in the system's interface name (see 1.3). Those parameters are:

- `mode<itf_index>`: this parameter changes the working mode of the interface between HPCAP mode (when the value is 2) and standard mode (if the value is 1). Note that an interface working in standard mode will not be able to fetch packets as interface in HPCAP mode would be able to, but the standard mode allows users to use for TX purposes (E.g.: launching a `scp` through this interface). An interface working in standard mode will no be able to be benefited byt the usage of the `detect-Pro10G` versions that can be found in the `samples` subfolder.
- `core<itf_index>`: this parameter fixes the processor core of the machine that will poll the interface for new packets. As this poll process will use the 100% of this CPU, affinity issues must be taken into account when executing more applications, such as detect-Pro10G. For further information see C.
- `vel<itf_index>`: this parameter allows the user to force the link speed that will be negotiated for each interface. Allowed values are 1000 and 10000.
- `caplen<itf_index>`: this parameter sets the maximum amount of bytes that the driver will fetch from the NIC for each incoming packet.
- `pages<itf_index>`: amount of kernel pages to be assigned to this interface's kernel-level buffer. The installation script will check the amount of pages to be used and make sure the sum of the pages used by all interfaces in HPCAP mode is the total. If this condition is not met, the installation script will issue an error message with useful information for changing this configuration.

Warning: changing any of the above mentioned parameters will take no effect until the driver is re-installed.

Once all the configuration parameters have been properly set, the execution of the script `install_hpcap.bash` will take charge of all the installation steps that need to be made in order to install the HPCAP driver.

1.3 Interface naming and numbering

This version of the driver allows choosing whether each interface is to work in HPCAP or standard (traditional, `ixgbe`-alike) modes. Consequently, a decision regarding the naming policy for those interfaces was made.

The target of this policy was always being able to identify each of the interfaces of our system regardless the mode it is working on (so you will be able to know which `<param_name><itf_index>` of the `params.cfg` file maps to each interface). This leaded to each interface being named as `<mode_identifyer><interface_index>`, where:

- `<mode_identifyer>`: can be `hpcap` when the interface is working in the HPCAP mode, or `xgb` if the interface works in standard mode.
- `<interface_index>`: this number will always identify the same interface regardless its working mode. For example, if the first interface found in the system is told to work in standard mode and second interface in the HPCAP mode, you will see that your system has the `xgb0` and `hpcap1` interfaces. If you revert the working mode for such interfaces you will then find interfaces named `hpcap0` and `xgb1`.

1.4 Per-interface monitored data

Once the driver has been properly installed, a monitoring script will be launched for each of the interfaces specified in the `ifs` configuration parameter. The data generated by those monitoring scripts can be found in the `data` subfolder. In order to avoid the generation of single huge files, the data is stored in subfolders whose names follow the format `<year>-<week number of year>`.

Inside each of those `<year>-<week number of year>` subfolders, you will find a file for each of the active interface plus a file with CPU and memory consumption related information.

On the one hand, the fields appearing in each of the `hpcapX` files are:

```
<timestamp> <RX-bps> <lost-bps-estimate> <RX-pps> <lost-pps>
```

On the other hand, the fields of the `cpus` file are:

```
<timestamp> <%-of-used-CPU(one for each CPU)> <total-memory> <used-memory> <free-memory>
 <cached-memory> <detect-pro-memory-consumption(one for each instance, 0 if none)>
```

1.5 Waking up an interface in *standard* mode

If a interface configured to work in standard mode wants to be configured, the `wake_standard_iface.bash` script is provided. Its usage is the following:

```
./wake_standard_iface.bash <iface> <ip addr> <netmask> <core> [<speed, default 10000>]
```

Where:

- `iface` is the interface you want to wake up. As this is thought for interfaces working in standard mode, the interface should be some `xgbN`.
- `<ip addr> <netmask>` are the parameters needed to assign an IP address and a network mask for this interface.
- `core` is the processor where all the interrupts regarding this will be sent, so we can make sure that such interrupts do not affect the capture performance of an HPCAP interface.
- `speed` is an optional parameter that allows you to force the link speed of this interface. Its values can be 1000 or 10000.

1.6 Sample applications

1.6.1 hpcapdd

`hpcapdd` is a sample program that maps HPCAP's kernel packet buffer for a certain interface and write its contents into the desired destination directory. Note that, in order to obtain maximum performance, the data access is made in a byte-block basis. This byte-block access policy has consequences regarding its usability, as it must be assured that the application starts running in a correct state. `hpcapdd` will generate data files following the RAW format (see 2).

`hpcapdd` has been programmed so it preformas an orderly close when receiving a `SIGINT` signal, so it must be ended with:

```
kill -s SIGINT ...
```

or

```
killall -s SIGIN ...
```

Importantly, **the HPCAP driver must be re-installed** before launching a new instance of `hpcapdd`.

1.6.2 hpcapdd_p

`hpcapdd_p` is a sample program that maps HPCAP's kernel packet buffer for a certain interface and write its contents into the desired destination directory. Note that `hpcapdd_p` accesses the data in a per-packet rather than in a byte-block basis. This have an effect damaging the write throughput performance but may result of interest as it has some interesting usability effects. `hpcapdd_p` will generate data files following the RAW format (see 2).

`hpcapdd_p` has been programmed so it preformas an orderly close when receiving a `SIGINT` signal, so it must be ended with:

```
kill -s SIGINT ...
```

or

```
killall -s SIGIN ...
```

Differently from `hpcapdd`, `hpcapdd_p` does not require the HPCAP driver to be reinstalled before launching a new instance of the program.

Importantly, if an instance of `hpcapdd` is to be run after closing an instance of `hpcapdd_p`, the HPCAP drriver must be re-intalled. In the opposite case (launching an `hpcapdd` instance and then an `hpcapdd_p` one) no driver re-installation is needed.

2 Working with the RAW file format

This section describes the structure of the "raw" format files generated by the usage of the HPCAP driver as explained in [1].

RAW files are generated by the programs that fetch traffic from the network using the `HPCAP` driver (see 1.6).

2.1 File data structures

A raw file is composed by a set of consecutive packets. Each packet is preceded by its corresponding header which contains information related to the packet just as shown in Fig.1:

- **Seconds** 4 bytes containing the seconds field of the packet timestamp.
- **Nanoseconds** 4 bytes containing the nanoseconds field of the packet timestamp.
- **Caplen** 2 bytes containing the amount of bytes of the packet included in the file.
- **Len** 2 bytes containing the real size of the packet.

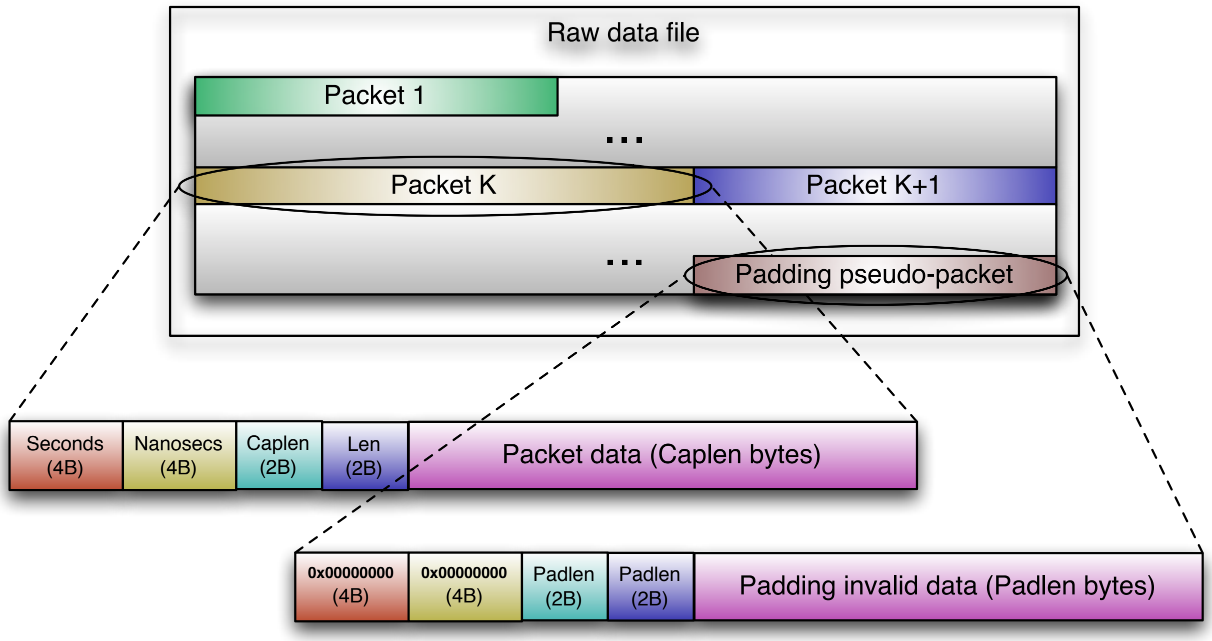

Figure 1: Raw file format

The end of the file is denoted by the appearance of a pseudo packet showing the amount of padding bytes added at the end of the file (in order to generate files of the same size). The padding pseudo-packet has a similar header than any other packet in the file with the difference that both the "Seconds" and the "Nanoseconds" fields are set to zeros. Once the padding pseudo-packet has been located, the padding size can be read from any of the "Len" or "Caplen" fields. Note that the padding length could be zero.

2.2 Example code

The next pages show an example code for a programs that reads a raw file and generates a new pcap file with the contents of the first one.

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <sched.h>
#include <unistd.h>
#include <assert.h>
#include <errno.h>
#include <sys/time.h>
#include <pcap/pcap.h>

#include "../../include/hpcap.h"
#include "raw2.h"

int main(int argc, char **argv)
{
    FILE* fraw;
    pcap_t* pcap_open=NULL;
    pcap_dumper_t* pcapture=NULL;
    u_char buf[4096];
    struct pcap_pkthdr h;
    u_int32_t secs,nsecs;
    u_int16_t len;
    u_int16_t caplen;
    int i=0,j=0,k=0,ret=0;
    char filename[100];
    u_int64_t filesize=0;

    if( argc != 3 )
    {
        printf("Uso: %s <fichero_RAW_de_entrada> <fichero_PCAP_de_salida>\n", argv[0]);
        exit(-1);
    }

    fraw=fopen(argv[1],"r");
    if( !fraw )
    {
        perror("fopen");
        exit(-1);
    }

    //abrir fichero de salida
    #ifdef DUMP_PCAP
        sprintf(filename,"%s_%d.pcap",argv[2],j);
        pcap_open=pcap_open_dead(DLT_EN10MB,CAPLEN);
        pcapture=pcap_dump_open(pcap_open,filename);
        if( !pcapture)
        {
            perror("Error in pcap_dump_open");
            exit(-1);
        }
    #endif

    while(1)
    {
        #ifdef PKT_LIMIT
        i=0;
        while( i<PKT_LIMIT )
        #endif
        {
            /* Lectura de info asociada a cada paquete */
```

```
if( fread(&secs,1,sizeof(u_int32_t),fraw)!=sizeof(u_int32_t) )
{
    printf("Segundos\n");
    break;
}
if( fread(&nsecs,1,sizeof(u_int32_t),fraw)!=sizeof(u_int32_t) )
{
    printf("Nanosegundos\n");
    break;
}
if( nsecs >= NSECS_PER_SEC )
{
    printf("Wrong NS value (file=%d,pkt=%d)\n",j,i);
    printf("[%09ld.%09ld] %u bytes (cap %d), %lu, %d,%d\n", secs, nsecs, len, caplen, filesize,j,i);
    //break;
}
if( (secs==0) && (nsecs==0) )
{
    fread(&caplen,1,sizeof(u_int16_t),fraw);
    fread(&len,1,sizeof(u_int16_t),fraw);
    if( len != caplen )
        printf("Wrong padding format [len=%d,caplen=%d]\n", len, caplen);
    else
        printf("Padding de %d bytes\n", caplen);
    break;
}

if( fread(&caplen,1,sizeof(u_int16_t),fraw)!=sizeof(u_int16_t) )
{
    printf("Caplen\n");
    break;
}
if( fread(&len,1,sizeof(u_int16_t),fraw)!=sizeof(u_int16_t) )
{
    printf("Longitud\n");
    break;
}

/* Escritura de cabecera */
h.ts.tv_sec=secs;
h.ts.tv_usec=nsecs/1000;
h.caplen=caplen;
h.len=len;

#ifdef DEBUG
printf("[%09ld.%09ld] %u bytes (cap %d), %lu, %d,%d\n", secs, nsecs, len, caplen, filesize,j,i);
#endif
if( caplen > MAX_PACKET_LEN )
{
    break;
}
else
{
    /* Lectura del paquete */
    if( caplen > 0 )
    {
        memset(buf,0,MAX_PACKET_LEN);
        ret = fread(buf,1,caplen,fraw);
        if( ret != caplen )
        {
            printf("Lectura del paquete\n");
            break;
        }
        #ifdef DEBUG2
        for(k=0;k<caplen;k+=8)
        {
```

```
                        printf( "\t%02x %02x %02x %02x\t%02x %02x %02x %02x\n",
                            buf[k], buf[k+1], buf[k+2], buf[k+3],
                            buf[k+4], buf[k+5], buf[k+6], buf[k+7]);
                    }
                    #endif
                }
            }
            /* Escribir a fichero */
            #ifdef DUMP_PCAP
                pcap_dump( (u_char*)pcapture, &h, buf);
            #endif
            i++;
            filesize += sizeof(u_int32_t)*3+len;
        }

        #ifdef PKT_LIMIT
            printf("%d paquetes leidos\n",i);
            if(i<PKT_LIMIT)
                break;
            j++;
            #ifdef DUMP_PCAP
                pcap_dump_close(pcapture);
                //abrir nuevo fichero de salida
                sprintf(filename,"%s_%d.pcap",argv[2],j);
                pcap_open=pcap_open_dead(DLT_EN10MB,CAPLEN);
                pcapture=pcap_dump_open(pcap_open,filename);
                if( !pcapture)
                {
                    perror("Error in pcap_dump_open");
                    exit(-1);
                }
            #endif
        #endif
    }
    printf("out\n");

    #ifndef PKT_LIMIT
        #ifdef DUMP_PCAP
            pcap_dump_close(pcapture);
        #endif
        j++;
        printf("%d paquetes leidos\n",i);
    #endif

    printf("%d ficheros generados\n",j);
    fclose(fraw);

    return 0;
}
```

A Quick start guide

This section shows how to properly install `HPCAP` in your system. We will assume you already have a `HPCAPX.tar.gz` file in your system, and we will show how to continue from there.

1. Check that your kernel is compatible with `HPCAP`. Nowadays `HPCAP` has been tested with `2.6.32`, `3.2`, `3.5` and `3.8` kernels.

2. Save your current network interface mapping so you can easily identify (using the MAC address) which interface is connected to which link:

   ```
   ifconfig -a > old_interfaces.txt
   ```

3. Decompress the contents of the data file.

   ```
   tar xzvf HPCAPX.tar.gz
   ```

   This command will create the `HPCAPX` directory in your system.

4. Enter the `HPCAPX` directory.

   ```
   cd HPCAPX
   ```

5. Edit the `params.cfg` file according to your current configuration[1] (see 1.2).

6. Install the driver using the installation script (you will need superuser privileges). The script will compile both the driver ans the user-level library if they have not been compiled before.

   ```
   ./install_hpcap.bash
   ```

7. Check that the monitoring script is on by checking the contents of the data files:

   ```
   tail -f data/<year>-<week/hpcapX
   ```

   Note that traffic counters will not be valid until there is at least one application fetching data from the corresponding (ifterface,queue) pair.

A.1 Launching `hpcapdd`

Once you have properly installed `HPCAP` on your system, you can use `hpcapdd` (or similar programs) to store the traffic from the network into your system.

1. Make sure you have enough space for traffic storage. it is recommended to use a different volume than the used for your operating system. You can check by executing

[1] In order to identify the NICs you may need to launch the installation script once in order to use the MAC to identify which interfaces are to work on `HPCAP` or standard mode, the re-edit the `params.cfg` file and install the driver using the script again.

```
df -h
```

2. Go to the `hpcapdd` directory (assuming we are already at the `HPCAPX` directory).

```
cd samples/hpcapdd
```

3. Launch the application (you will need superuser privileges)

`taskset -c 3 ./hpcapdd 3 0 /storage` [2].

A.2 Checking traffic storage

1. Check the counter files in the `data` subdirectory.

2. In your storage target directory, list the files that have already been written

```
ls -l /storage/*
```

All of the generated files should have a size of 2GB = 2147483648bytes.

3. Convert one file from `raw` to `pcap`

```
cd HPCAPX/samples/raw2
./raw2 /storage/<dir>/<raw_file> <pcap_file>
```

If the capture is being properly made, the program should end showing the message:

```
Padding de XX bytes
```

with XX being the amount bytes added at the end of the file for obaining a file size multiple of the filesystem's block size.

[2]In this case the application will fetch the traffic arriving to the queue 0 on `hpcap3`. The core 1 has been chosen in order to not interfere with the kernel-level capture thread and to avoid interception from different NICs (assuming a configuration as the one shown in B)

B Configuration example

Here you can find an example of `params.cfg` file taken from a real `HPCAP` installation.

```
########################################################################
#
# HPCAP's configuration file
#
########################################################################

###################
# HPCAP location
basedir=/home/naudit/HPCAP4
###################

###################
# Driver version
version=hpcap_ixgbe-3.7.17_buffer;
###################

###################
# Number of RX queues
nrxq=1;
###################

###################
# Number of TX queues
ntxq=1;
###################

###################
# Interface list
#  E.g.:
#    ifs=hpcap0 xgb1 hpcap2 ...;
ifs=hpcap0 hpcap3;
###################

###################
# Total number of interfaces in the system (ussually 2 times the number of cards)
nif=4;
###################

###################
# Mode
#       1 = standard ixgbe mode (interface name = "xgb[N]")
#       2 = high performance RX (interface name = "hpcap[N]")
# E.g.:
#       mode0=1; <---- xgb0 will be set to standard ixgbe mode
#       mode0=2; <---- hpcap0 will be set to high performance RX
#       mode1=2; <---- hpcap1 will be set to high performance RX
mode0=2;
mode1=1;
mode2=1;
mode3=2;
###################

###################
# Core to start mapping threads at
#    run "numactl --hardware" to check available nodes
#    a value of -1 means the last core
#    a value greater or equal than zero means the specified core
# E.g.:
#    core0=3 <---- hpcap0 (or xgb0) queues will be mapped from core 3 and so on
#    core2=6 <---- hpcap2 (or xgb2) queues will be mapped from core 6 and so on
core0=0;
```

```
core1=-1;
core2=-1;
core3=2;
###################

###################
# Dup
#       0 = do not remove duplicated packets
#       1 = remove duplicated packets
# E.g.:
#       dup0=1; <---- hpcap0 will remove duplicate packets
#       dup1=0; <---- hpcap1 will not remove duplicates packets
dup0=1;
dup1=1;
dup2=0;
dup3=0;
###################

###################
# Link speed
#    1000 = 1 Gbps
#    10000 = 10 Gbps
# E.g.:
#       vel0=1000;  <---- hpcap0 will be negotiated at 1Gbps
#       vel3=10000; <---- hpcap3 will be negotiated at 10Gbps
vel0=10000;
vel1=1000;
vel2=1000;
vel3=10000;
###################

###################
# Packet capture Length (caplen)
#    0 = full packet
#    x>0 = first x bytes
# E.g.:
#       caplen0=60; <---- only the 60 first bytes for packets coming through hpcap0 will be captured
#       caplen1=0;  <---- the full packet will be captures for traffic coming through hpcap1
caplen0=0;
caplen1=0;
caplen2=0;
caplen3=0;
###################

###################
# Number of pages for each interfaces's kernel buffer
# Total amount of pages: HPCAP_BUF_SIZE/PAGESIZE
#        echo £(( £(( £(cat /home/naudit/HPCAP4/include/hpcap.h | grep HPCAP_BUF_SIZE | tail -n 1 | awk '{print £4
# The minimal value for an interface in hpcap mode is 1
#
# E.g.:
#       pages0=1; <---- 1 page for hpcap0's kernel buffer
#       pages1=1024;  <---- 1024 pages for hpcap1's kernel buffer
#
# with 4MByte pages:
#    1 GB = 262144 pages
#    512 MB = 131072
pages0=131072;
pages1=1;
pages2=1;
pages3=131072;
###################

###################
# Monitor script sampling interval (seconds)
#    must be >0
```

```
monitor_interval=5;
###################

###################
# Core on which monitoring scripts will be executed on
#    must be >= -1
#    a value of -1 means the last core
#    a value greater or equal than zero means the specified core
monitor_core=-1;
###################

######################################################################
######################################################################
```

C Frequently asked questions

- **Which Linux kernel versions does HPCAP support?**

  HPCAP has been written for working under diverse Linux kernel/distribution combinations. Specifically, it has been tested for the following versions:

  - **OpenSuse:** 2.6.32 .
  - **Ubuntu/Debian:** 2.6.32, 3.2.0, 3.13.0 .
  - **Fedora:** 3.5.3, 3.14.7 .

- **Should I always use all the available interfaces in `hpcap` mode?**

  No. The reason for this is that the total amount of memory that this driver can use as internal buffer is 1GB, and this amount is divided between the `hpcapX` interfaces present in your system. Thus, if you are not going to capture packets from one interface configure it to work in standard mode and you will have bigger buffers for your capturing interfaces. The amount of memory assigned to each interface out of this 1GB is configured via the `pages` parameter (see 1.2).

- **Can I use more than one RX queue?**

  If you are going to use Detect-Pro the answer is no. The current version of Detect-Pro support packet capture for just one RX queue. Otherwise you can use more than one RX queue by changing `nrxq` parameter in the `params.cfg` file. Notice that if you use more than one queue per interface you will need to instantiate several packet-consuming applications (at least one per queue) in order to fetch all the data).

- **Can I simultaneously capture data from an interface with `dd` and `hpcapdd`?**

  Yes. In fact, you can use any amount of `dd` or `hpcapdd` instances over the same pair (interface,queue). The limit on the amount of simultaneous applications fetching data from the same pair (interface,queue) is defined in the `include/hpcap.h` file with the `MAX_LISTENERS` constant. Note that a change in this value will take no effect if the driver is not recompiled and re-installed in your system.

- **Is there a way to make sure that my system and user processes will not interfere in the capture process?**

Yes. First of all, you must make sure of which CPUs you are going to use for the HPCAP driver and Detect-Pro. At this point we have a list of CPUs that we want to isolate from the system scheduler. Now, depending on the distribution we are using:

- **OpenSuse:** you have to edit the `/boot/grub/menu.lst` file and add into the boot desired command line the following: `isolcpus=0,1,2,..,k` (with `0,1,2,..,k` are the CPUs to be isolated). The next time you boot your system your changes will have taken effect.
- **Ubuntu/Debian:** you have to edit the `/boot/grub/menu.lst` file, and in the `GRUB_CMDLINE_LINUX_DEFAULT` parameter add the following: `isolcpus=0,1,2,..,k`. Then, execute `update-grub` and the next time you boot your system your changes will have taken effect.

- **I am not obtaining the expected network capture performance, what is happening**
  Make sure that you have properly set the processor affinity for your storage programs (`dd`, `hpcapdd`) via the `taskset` command. You should check that the assigned processor is not used by any kernel capture thread (the ones specifies by the `coreX` parameters in the `params.cfg` file, see ??).

  Furthermore, you might be obtaining a poor performance because the kernel capture threads are not assigned to the same NUMA node where NIC is connected. In order to check that you should see the content of the file `/sys/bus/pci/devices/<pci_identifier>/numa_node` where `<pci_identifier>` can be obtained searching for your NIC in the `lspci` command's output. This file will tell you the NUMA node[3] you should schedule the capture threads for your NIC (a value of -1 means that there will be no performance difference regardless the NUMA node you use).

  If you're still obtaining a poor performance this may be due to the kernel-level memory buffer being stored in the opposite NUMA node than where the NIC is placed. In such cases you should keep the kernel capture thread in the NUMA node attached to your NIC (via the `coreX` parameter in `params.cfg` file), but should schedule your storage processes in the opposite NUMA node (via the `taskset` command). This way the inter-node memory transfers are minimized and maximum performance is met.

- **I see no traffic from the counters that appear on the `data/<year>-<week>` files**
  This is normal behaviour. Until there is at least one application listening, the kernel driver will not fetch packets and thus all the counters will be zero. Nevertheless, if the traffic intensity is very high, it is possible that the lost counter will be incremented but the amount is not to be trusted. This is a known issue that has not been solved yet.

- **Is it possible the size of the RAW files generated by `hpcapdd` or Detect-Pro?**
  Yes, you can. Remember the default file size is 2GB, which was chosen as a tradeoff between write performance ans data accesibility. However, if this file size does not fit your requirements you can change it (altough it is not reccommended due to the delicate operations involved) by editing the following parameter in the `HPCAPX/include/hpcap.h` file:

[3]Using the `numactl --hardware` command you can check which processors belong to each NUMA node.

```
#define HPCAP_BS ( 1048576ul )
#define HPCAP_COUNT ( 2048ul )
#define HPCAP_FILESIZE (HPCAP_BS*HPCAP_COUNT)
```

Changing the block size (`HPCAP_BS` parameter) is discouraged, so the right way of changing the generated files' size is changing the `HPCAP_COUNT` parameter.

Important: after changing these parameters (or anyone in the `hpcap.h` header file, it is required to re-compile the driver and any of the sample applications based on HPCAP used.

- **Is there a way to know which NUMA node is my NIC connected to?**

  Yes. The easiest way of obtaining this information is via the `lstopo` Linux command (which requires the installation of the `hwloc` package. An example of this command is:

```
$ lstopo
Machine (128GB)
  NUMANode L#0 (P#0 64GB)
    Socket L#0 + L3 L#0 (15MB)
      L2 L#0 (256KB) + L1 L#0 (32KB) + Core L#0 + PU L#0 (P#0)
      L2 L#1 (256KB) + L1 L#1 (32KB) + Core L#1 + PU L#1 (P#1)
      L2 L#2 (256KB) + L1 L#2 (32KB) + Core L#2 + PU L#2 (P#2)
      L2 L#3 (256KB) + L1 L#3 (32KB) + Core L#3 + PU L#3 (P#3)
      L2 L#4 (256KB) + L1 L#4 (32KB) + Core L#4 + PU L#4 (P#4)
      L2 L#5 (256KB) + L1 L#5 (32KB) + Core L#5 + PU L#5 (P#5)
    HostBridge L#0
      PCIBridge
        PCI 8086:1521
          Net L#0 "eth0"
        PCI 8086:1521
          Net L#1 "eth2"
      PCIBridge
        PCI 8086:10fb
          Net L#2 "hpcap0"
        PCI 8086:10fb
          Net L#3 "xgb1"
      PCIBridge
        PCI 8086:1d6b
      PCIBridge
        PCI 102b:0532
      PCI 8086:1d02
  NUMANode L#1 (P#1 64GB)
    Socket L#1 + L3 L#1 (15MB)
      L2 L#6 (256KB) + L1 L#6 (32KB) + Core L#6 + PU L#6 (P#6)
      L2 L#7 (256KB) + L1 L#7 (32KB) + Core L#7 + PU L#7 (P#7)
      L2 L#8 (256KB) + L1 L#8 (32KB) + Core L#8 + PU L#8 (P#8)
      L2 L#9 (256KB) + L1 L#9 (32KB) + Core L#9 + PU L#9 (P#9)
```

```
            L2 L#10 (256KB) + L1 L#10 (32KB) + Core L#10 + PU L#10 (P#10)
            L2 L#11 (256KB) + L1 L#11 (32KB) + Core L#11 + PU L#11 (P#11)
          HostBridge L#5
            PCIBridge
              PCI 1000:005b
                Block L#4 "sda"
```

Listing 1: Obtaining NUMA information using `lstopo`

In this example both NICs are connected to NUMA node 0.

Other way of getting this information is by means of the `lspci` command and the `/sys/` system's interface. An example of this is the following:

```
$ lspci
...
05:00.0 Ethernet controller: Intel Corporation 82599EB 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
05:00.1 Ethernet controller: Intel Corporation 82599EB 10-Gigabit SFI/SFP+ Network
Connection (rev 01)
...

$ cat /sys/bus/pci/devices/0000\:05\:00.0/numa_node
1
$ cat /sys/bus/pci/devices/0000\:05\:00.1/numa_node
1
```

Listing 2: Obtaining NUMA information using `lscpi` and `sysfs`

In this exampleboth NICs are connected to NUMA node 1.

Nota that this procedures are also valid for obtaining NUMA-related information for other PCI devices (i.e., an RAIC controller card, etc.).

References

[1] V. Moreno. Development and evaluation of a low-cost scalable architecture for network traffic capture and storage for 10gbps networks. Master's thesis, Escuela Politecnica Superior UAM, 2012.