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

User guide

HPCAP and Detect-Pro 10G

Víctor Moreno Martínez, Pedro M. Santiago del Río
victor.moreno@uam.es, pedro.santiago@uam.es

Madrid, 2013

Contents

1 Introduction

The HPCAP driver contains a series modifications made to the Intel's `ixgbe` driver [1] in order to achieve line-rate packet capture and storage for 10 Gb/s networks. This document will help you when installing and making use of such driver, but if you are looking for a further explanation on how does the driver work, you should take a look at [2].

2 Using the HPCAP driver

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

```
HPCAPX
|--> deps
|    |--> package files to be installed for debian-base distros
|--> doc
|    |--> documentation files
|--> driver
|    |--> hpcap_ixgbe-3.7.17_buffer
|    |    |--> driver
|    |    |  |-> driver source code files
|--> include
|    |-> common files for both driver and user level apps
|--> lib
|    |--> library for user apps source files
|--> samples
|    |--> bufdump
|    |--> example program that dumps the content of an hpcap buffer
|    |--> detectpro-10G
|    |    |--> bin
|    |    |  |--> source code for detect-pro
|    |    |--> data
|    |--> hpcapdd
|    |    |--> example program mapping hpcap's buffer driver and storing packets into hard-disk
|    |--> killwait
|    |    |--> application that allows a program that has been locked waiting for data from a
|    |    |   hpcap buffer to terminate
|    |    |--> raw2
|    |    |    |--> RAW file format to PCAP conversion example
|--> install_hpcap.bash
|--> params.cfg
|--> scripts
|    |--> scripts used to monitor and configure hpcap driver
```

2.1 Installing all the required packages

Inside the `deps` folder you will find a `install.bash` script that will install the required packages in a Debian-based distribution. 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.

2.2 Configuring 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`.

- **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). Check 2.3 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 2.3). Those parameters are:

  - **modeX**: 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` sub-folder.

  - **coreX**: 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 5.2. A value of -1 will assign the last core in the system.

  - **velX**: this parameter allows the user to force the link speed that will be negotiated for each interface. Allowed values are 1000 and 10000.

  - **caplenX**: this parameter sets the maximum amount of bytes that the driver will fetch from the NIC for each incoming packet.

- **dupX**: this parameter enables/disables (1=enable, 0=disable) the removal of L1 duplicated packets on each interface. This option is hard-code disabled. If you want to enable it you have to edit the file `include/hpcap.h` ann uncomment the line containing: `#define REMOVE_DUPS`.

- **Monitor interval and core**: those parameters control how often (in seconds) the monitor scripts will poll the interfaces for data, and the core those monitoring scripts will be executed at (a value of -1 refers to the last core in the system).

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.

Warning:	changing any of the above mentioned parameters will take no effect until the next time the driver is installed (i.e. the `install_hpcap.bash` is executed)

2.3 Interface naming and numbering

This version of the driver allows choosing whether each interface will work in HPCAP or standard (traditional, `ixgbe`-alike) modes. Consequently, an interface naming policy had to be defined.

The goal is 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>` on 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`.

2.4 Per-interface monitored data

Once the driver has been properly installed, a monitoring script will be launched for each `hpcapX` interface specified in the `ifs` configuration parameter. The data generated by those monitoring scripts can be found in the `data` sub-folder. In order to avoid the generation of single huge files, the data is stored in sub-folders 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> <% CPU used (per CPU)> <total memory> <used memory> <free memory> <cached memory>`
`<detPro memory consumption (per instance, 0 if none)>`

Some of the monitoring scripts parameters can be configured via the `params.cfg` file (see 2.2).

2.5 Using an interface in *standard* mode

If the user needs an interface working in standard mode to be woken up, it must be included in the `ifs` list inside the `params.cfg` file.

Once the interface is woken up, it can be used as a standard network interface (e.g. use `ifconfig`, `ethtool`, `tcpdump` , ...).

Warning:	if a standard interface is not woken up by being included in the `ifs` parameter, its usage may damage capture performance for other interfaces due to processor affinity issues

3 Capturing network data with *dd*

The goal of `HPCAP` is enabling to capture and store network traffic at high rates. To accomplish such goal the datapath has been optimized to make most of the disk (or other non-volatile storage) write performance. In particular, once the packets arrive to the NIC, the driver makes them available on a byte-stream basis (see 6).

This allows the user to use standard programs such as `dd` to move the byte-stream from the driver space into the target device. If a user wants to make use of that feature, he should execute:

```
dd if=/dev/hpcap_X_Q of=<path-to-target-file> bs=1M count=2048 oflag=direct
```

With `X` the index of the interface to capture the traffic from and `Q` the corresponding queue (by default there is just one RX queue so this number will be 0, but there is support for more than one queue).

This will generate $2GB$-sized files whose content will be written in RAW format (see 6).

Warning:	if you use `dd` to capture traffic and you kill the capture process before a 2GB file has been completed, you will need to restart the driver if you want to capture traffic from this interface again.

Warning:	if you use `dd` to capture traffic and suddenly no traffic is received, the `dd` process will get stuck at a driver lock and you will need to use the `killwait`[0] sample program to solve this situation. The previous warning also applies in this situation.

Inside the `scripts` folder you can find the `copy.bash` which contains an example of how to use the `dd` program to fetch the incoming network data. Note that the use of the `dd` must be made taking into account processor affinity issues: the program must be scheduled to be executed to not interfere with one of the processors used for packet capture (specified in the `params.cfg` file, as explained in 2.2). This is easily set by using the UNIX `taskset` command preceding the `dd` call.

[0]The `killwait` program can be found in the `samples` directory. If you execute this program with no arguments you will obtain the execution help.

4 Capturing network data with *hpcapdd*

The `hpcapdd` program was created as an improved version of the `dd` program for being used to fetch network data from an `HPCAP` interface. The distinguishing features of `hpcapdd` are:

- it directly maps the kernel memory buffer (see [2]) into user space to eliminate intermediate copies.
- the way the network data is accessed includes a timeout, so if no more packets are received by the associated interface the user can stop the `hpcapdd` instance by sending the `SIGKILL` signal to the process.

4.1 Launching and stopping network traffic capture with `hpcapdd`

The use of `hpcapdd` is quite simple. The program can be found in the `samples/hpcapdd` folder, and its execution is as follows:

```
./hpcapdd <interface index> <queue index> <target directory>
```

Where:

- `interface index` is the number identifying the interface to capture traffic from. For example this would be a 2 if capturing from `hpcap2`, a 3 for `hpcap3` and so on. It is necessary that the index belongs to a network interface working in `HPCAP` mode.
- `queue index` is the number of queue of the interface to capture from. As explained in 6.2 the default number of queues is 1, so this value should be 0 in that case, but there is compatibility for a higher number of RX queues.
- `target directory` is the path (absolute or relative) to the directory where the network data is to be stored. This directory should belong to a device with a write throughput high enough in order to avoid bottlenecks ant hus packet loss (e.g. RAID or SSD volumes). See 4.2 to learn about how will the data be stored in such directory. If this argument is set to the string "`null`", no data will be written (`hpcapdd` becomes a dummy packet-consumer program).

As it has been previously said, any `hpcapdd` instance can be killed anytime by sending the corresponding process the `SIGKILL` signal.

Warning:	if you use `hpcapdd` to capture traffic and you kill the capture process before a 2GB file has been completed, you will need to restart the driver if you want to capture traffic from this interface again.

4.2 Data storage structure

When a network data storage process is launched via the `hpcapdd` program, the data will be stored inside the `<target directory>` specified when invoking the program. As previously stated, the data will be stored in $2GB$ files following the RAW file format (see 6).

The data files generated by `hpcapdd` have the following name structure:

`<begin timestamp>_<interface>_<queue>.raw`.

Those data files are stored in subdirectories separating every 30 minutes of traffic (and named with the timestamp of the beginning of such 30-min period). Inside each directory you will find all the data files whose first packet belong to such 30-min period.

Note that if the incoming network traffic has a low rate, one data file could cover more than one 30-min period. In such cases the next data file will be stored in its corresponding subdirectory, giving the impression that there are 30-min period missing.

An example of how data is stored is the target directory is the following:

```
/storage/
|-- 1391720400
|   |-- ...
|   |-- 1391723988_hpcap3_0.raw
|   |-- 1391723990_hpcap3_0.raw
|   |-- 1391723993_hpcap3_0.raw
|   |-- 1391723995_hpcap3_0.raw
|   |-- 1391723997_hpcap3_0.raw
|   |-- 1391723999_hpcap3_0.raw
|-- 1391722200
|   |-- 1391724001_hpcap3_0.raw
|   |-- 1391724003_hpcap3_0.raw
|   |-- 1391724006_hpcap3_0.raw
|   |-- 1391724008_hpcap3_0.raw
|   |-- ...
...
```

4.3 hpcapdd_p5g

The hpcapdd program reads data from the network interface in a byte-stream basis, ignoring the network packet semantics of the data. For such reason, if one hpcapdd process is stopped when a $2GB$ file has not been completely generated a new hpcapdd process would begin where the previous ended breaking the RAW file format. For this reason, the driver must be re-installed before starting a new hpcapdd process.

In order to eliminate this limitation, a program that reads the network data and stores it in a per-packet basis has been created. The name of this program is hpcapdd_p5g (where the "p" stands for the per-packet basis and the "5g" stands for the achieved rate).

The execution of hpcapdd_p5g has the same arguments as hpcapdd and the same output behaviour.

Note that when using hpcapdd_p5g:

- you will only be able to capture traffic up to $5Gbps$ (half of the rate obtained with hpcapdd)
- you won't need to re-install the driver for launching a new hpcapdd_p5g after having killed a previous one

4.4

5 Detect-Pro 10G

In the current version, the `detectpro-10G` folder is located folder inside the `HPCAPX/hpcap_ixgbe-3.7.17_buffer/samples` subfolder. You will also find a `detectpro-10G-bonding` version that allows to process packets from several interfaces and merge them as one.

5.1 Detec-Pro 10G and threads

Each instance of Detect-Pro to be executed in your system will create the following amount of threads:

- **Main thread:** this thread is the one that creates the rest of threads and is in charge of receiving stop signals from the user.
- **Dump thread:** this thread is fed by the packets captured by the HPCAP driver captures and writes them into non-volatile storage. The format of the files generated is the RAW format (the one used by the driver to store the packets in its buffers, see).
- **Process thread:** this thread is fed by the packets captured by the HPCAP driver captures and processes them in order to create one-directional flow records.
- **Export thread:** this thread exports the flow records created by the process thread once they have been close and stores them into non-volatile storage.

5.2 Affinity issues

As each instance of Detect-Pro creates several threads, CPU affinity must be carefully planned. This is done by modifying the values of the following parameters (in the `global.cfg` file inside the Detect-Pro folder:

- `affinity_dump`
- `affinity_process`
- `affinity_export`
- `affinity_main`

In order to avoid interferences from one thread into another, they all should be mapped into different CPUs. Take into account that the CPUs that have already been used by the HPCAP driver cannot be used by Detect-Pro.

6 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 [2].

RAW files are generated by the programs that fetch traffic from the network using the `HPCAP` driver (see 3, 4).

6.1 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.

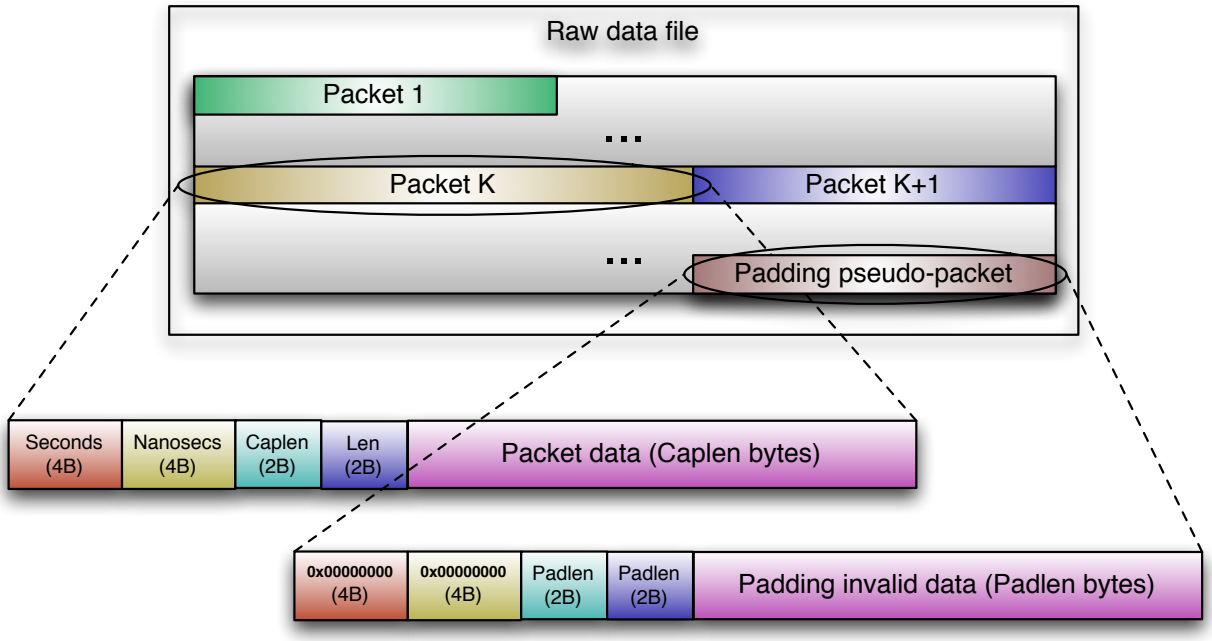

Figure 1: Raw file format

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.

6.2 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.

```
#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);
    //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;
}
```

Frequently asked questions

- **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.

- **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:

  - **Suse:** 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 2.2).

  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[1] 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.

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

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[2] (see 2.2).

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

[2] 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 1 ./hpcapdd 3 0 /storage
```
[3].

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
```

[3] In this case the application will fetch the traffic arriving to the queue 0 on `hpcap2`. 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=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 standard ixgbe mode
#       mode1=2; <---- hpcap1 will be set to high performance RX
mode0=1;
mode1=1;
mode2=2;
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=-1;
```

```
core1=-1;
core2=-1;
core3=0;
####################

####################
# 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=0;
dup1=0;
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=10000;
vel2=10000;
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;
####################

####################
# 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;
####################

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

References

[1] Intel. Intel 82599 10 GbE Controller Datasheet. *October*, (December), 2010.

[2] 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.