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 suche driver, but if you are looking for a further explanation on how does the driver work, you should check [2].

2 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:

```
HPCAPX
|-> deps
| |-> package files to be installed for debian-base distros
|-> doc
| |-> documentation files
|-> hpcap_ixgbe-3.7.17_buffer
| |-> changes.log
| |-> 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
|   | |-> global.cfg
|   | |-> networks.cfg
|   | |-> readme.txt
|   |-> detectpro-10G-bonding
|   | |-> bin
|   | |  -> source code for detect-pro with interface bonding support
|   | |-> global.cfg
|   | |-> interfaz
|   | | |-> detect-pro's web interface source code
|   | |-> networks.cfg
|   | |-> readme.txt
|   |-> 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
| |-> script used to monitor and configure hpcap driver
|-> wake_standard_iface.bash
```

2.1 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.

2.2 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`.
- `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:
  - `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 3.2.
  - `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.

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.

2.3 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.

2.4 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)>
```

2.5 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.

3 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.

3.1 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.

3.2 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`$_d ump$`affinity`$_p rocess$`affinity`$_e xport$`affinity`$_m ain$

  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.

4 Working with the RAW file format

sec:raw

This section describes the structure of the "raw" format files generated by the usage of the HPCAP driver as explained in [2].

4.1 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.

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.

4.2 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;
}
```

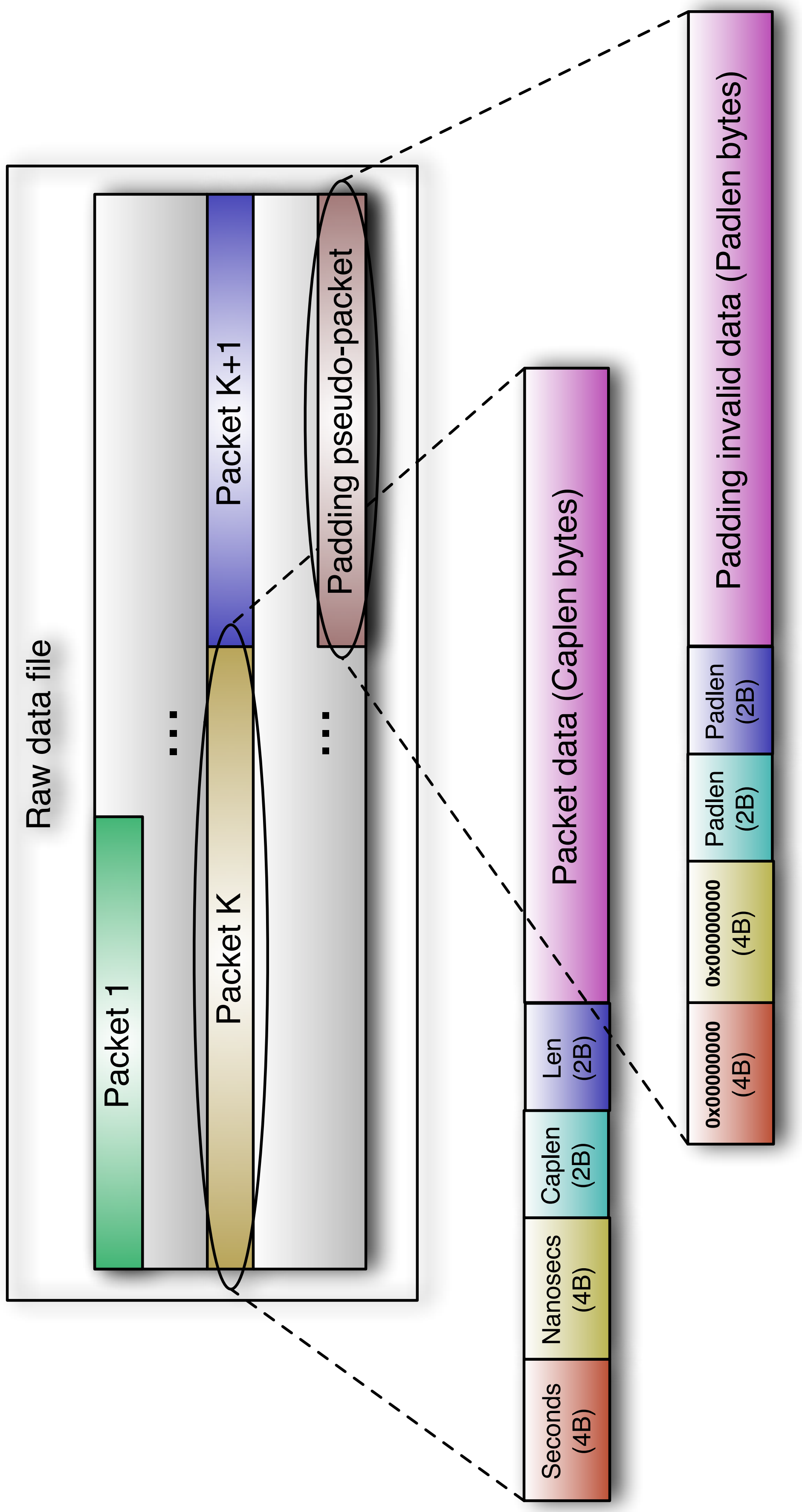

Figure 1: Raw file format

5 Frequently asked questions

- **Should I always use all the `hpcapX` interfaces available?**
  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.

- **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 we depending on the distro we are using:

  - **Suse:** we have to edit the `/boot/grub/menu.lst` file and add into the boot command line we desire the following: `isolcpus=1,2,..,n` where `1,2,..,n` are the CPUs to be isolated. The next time you boot your system your changes will have taken effect.
  - **Ubuntu:** we have to edit the `/boot/grub/menu.lst` file, and in the `GRUB_CMDLINE_LINUX_DEFAULT` parameter add the following: `isolcpus=1,2,..,n`. Then, execute `update-grub` and the next time you boot your system your changes will have taken effect.

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.