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, 2014

Contents

1 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 **??**, **??**).

1.1 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.

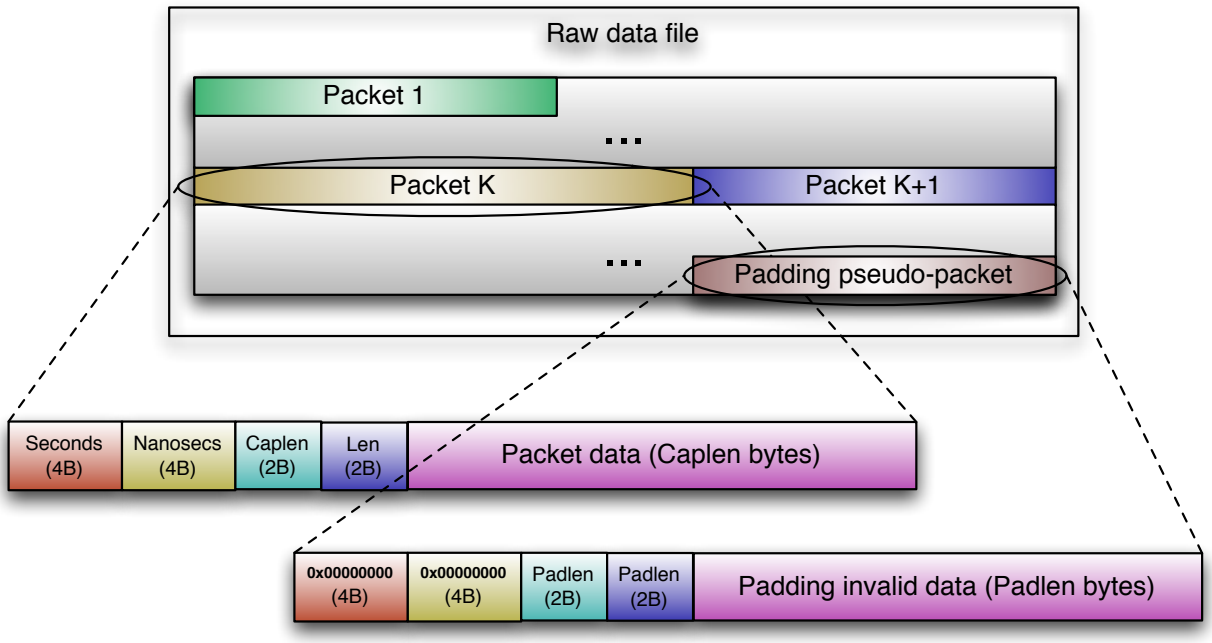

Figure 1: Raw file format

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.

1.2 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);
    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;
}
```

Frequently asked questions

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

  HPCAP has been written for working under diverse Linux kernel/distribution combinations. Namely, 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.

- **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[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 **??**).

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] 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.