HIGH PERFORMANCE COMPUTING AND NETWORKING RESEARCH GROUP

Universidad Autónoma de Madrid
Escuela Politécnica Superior

HPCAP

User Guide

Abstract

The HPCAP User Guide contains the basic instructions to use the HPCAP capture driver for high performance networks. It will show you how to install it, configure it according to your needs and run related programs. It also describes the RAW file format, the default output of HPCAP client applications.

Víctor Moreno Martínez, Guillermo Julián Moreno
victor.moreno@uam.es, guillermo.julian@estudiante.uam.es[1]

November 1, 2016

[1]Current maintainer.

How to read this manual

A short guide to get the HPCAP driver up and running without complications is presented in section 1. The next sections explain in more detail the HPCAP driver and how does it work: section 2 details the requirements of the driver and explains how to install HPCAP as a standard Linux driver, that is, allowing you to use standard system tools like *modprobe* to use HPCAP.

Section 3 details the configuration parameters possibilities of HPCAP. Section 3.3 is specially relevant, as it explains how to properly configure the core assignments in order to get the best performance. Section 4 explains how to capture traffic and monitor the performance of HPCAP. Finally, section 6 answers some questions that may pop up during the installation, usage and configuration of HPCAP. Refer to this section if you find any weird bug you can't solve: it may be solved there.

Contents

Chapter 1

Quick start guide

1.1 Driver installation

This section shows how to install HPCAP in your system. We will assume you already have the HPCAP sources decompressed in a *HPCAP* directory.

1. Check that your kernel is compatible with HPCAP. Nowadays HPCAP has been tested with 2.6.32, 3.2.0, 3.5.3, 3.11.0, 3.13.0, 3.14.7, 3.15.0, 3.19.0 kernels.
2. Install the required packages:

   CentOS : *yum install libpcap-devel numactl numactl-devel*
3. Configure your installation. The easiest way is to run `scripts/gen-hpcap-config params.cfg`, that will ask you some basic questions and will generate a basic configuration file, including the core assignments based on your NUMA topology.

   This script should be enough for most situations. However, for more control over the installation, you can manually copy the *params.cfg.sample* sample file to *params.cfg* and edit it. The sample configuration file has all the available parameters documented. For more details on the parameters in the file, more details can be found in section 3.1.
4. Install the driver using the installation script, running `sudo ./install_hpcap.bash` (you will need superuser privileges). The script will compile both the driver and the user-level library if they have not been compiled before.

For more details on driver installation procedures, including how to use HPCAP as a standard driver with `modprobe` and similar tools, see section 2.

1.2 Checking driver status

The driver provides several ways for the user to check the status. The most useful is the script *script/hpcap-status*, which will show a continuous screen with the essential information of the driver (reception rate, installation status and other errors, see figure 1.1).

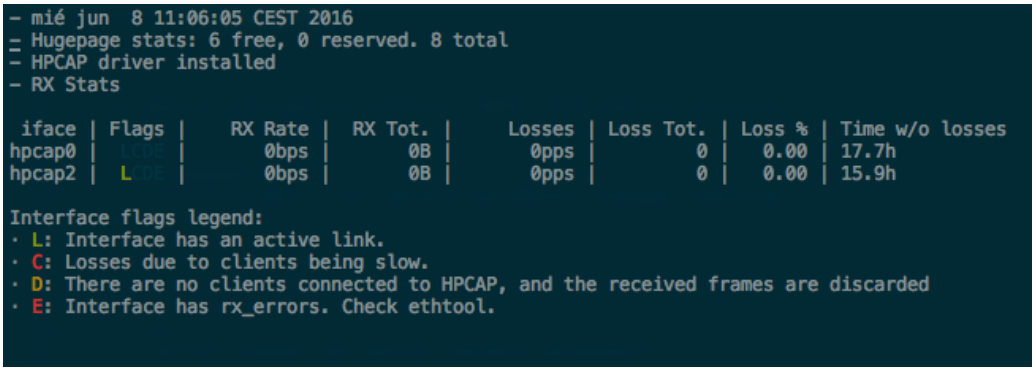

Figure 1.1: Script with the driver status. Each interface has statistics on the reception behaviour, and flags indicating its status.

The kernel log (*/var/log/kernel.log* and/or `dmesg`, depending on the Linux distribution) will contain the messages from the driver that will show any error that may have happened.

HPCAP also provides interface monitors that log the bytes and frames received and lost[1]. These logs are accessible in the directory *data/[year]-[week]*, with a file named for each interface. You can run `tail -f data/[year]-[week]/[interface]` to ensure that the monitor for that interface is running.

HPCAP integrates with the standard Linux networking tools, so commands like `ifconfig` or `ethtool` will be useful to check link status or the total losses of the interface.

Note: Both *hpcap-status* and the monitor log files will only show received traffic when there's an application, such as `hpcapdd`, listening for data in the driver.

1.3 Traffic capture: 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 the space on each disk by executing `df -h`.
2. Launch the application (you will need superuser privileges). Assuming you've run already `make` and are in the *HPCAP* directory, run `taskset -c 3 bin/release/hpcapdd 3 0 /storage .` In this case the application will fetch the traffic arriving to the queue 0 on `hpcap3`. The core 3 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 A).

1.4 Checking and reading captured traffic

1. Check the counter files in the `data` subdirectory to ensure no losses have ocurred. You can use the script *scripts/parse-hpcap-log*, that will summarize the log so you can tell easily the receive rate and whether losses ocurred during the capture. See section 4.2 for more details on the monitoring facilities of HPCAP.
2. In your storage target directory, list the files that have already been written with `ls -lh /storage/*`. All of the generated files should have a size of 2 GB = 2147483648 bytes.
3. Convert one file from `raw` to `pcap` running `bin/release/raw2pcap /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.

   You can also use the application `bin/release/chechraw <storage_dir>` to check that the RAW files are properly formed without converting anything. See section 4.1.3 for more details.

1.5 When something is not working right

Misconfigurations of the driver or unsual environments can cause the driver to perform badly or even crash. Some of these problems have easy workarounds:

- In some cases, the NIC may be failing to recognize certain VLAN encapsulations, and will throw frames where the unrecognized encapsulation plus the payload exceeds the card's MTU. The solution is to increase the MTU to accept those unrecognized frames: `ifconfig hpcapX mtu 1522` (or more bytes if necessary).
- If the link has heavy traffic, HPCAP may fail to negotiate the speed and bring the link up, showing no Ethernet level link. The solution is to manually set the link speed with the comand `ethtool -s <iface> speed <link-speed> duplex full autoneg off`, where `link-speed` is the speed in Mbps (usually 10000).
- Some applications and features may interfere and decrease HPCAP performance. Ensure the irqbalance service is deactivated and HyperThreading switched off.
- Ensure that the driver's poll threads are running on the same NUMA node of the NIC card. See section 3.3 for details.

[1] The monitor interval is configurable in the *params.cfg* file

For other bugs and known errors, read section 6. If the error persist, remember that the script *scripts/diagnostics* will collect all necessary information from the system and pack it in a compressed file, so it can be sent to the developers for debugging.

Chapter 2

Installation

The basic installation procedure is explained in section 1.1. Here you can find more details on the requirements and installation modes.

2.1 Requirements

HPCAP is distributed as a source code, so you will need the build tools (`make` and `gcc`, mainly) and the Linux headers in order to compile the driver. Apart from this, you will also need some libraries and tools:

- **libpcap**: Both the library and the development headers. It's used for the samples that read or write to *pcap* files.
- **numactl**: Includes binaries, static library and the development headers. The binaries are used to detect NUMA architectures and configure the driver automatically. The library and headers are used to get the same information about NUMA architectures in some sample applications.

Apart from this, HPCAP may not build or run correctly with unsupported Linux kernel versions. HPCAP has been tested with the following versions: 2.6.32, 3.2.0, 3.5.3, 3.11.0, 3.13.0, 3.14.7, 3.15.0, 3.19.0.

Kernels other than the ones listed may introduce changes that can break the driver compilation, or make it fail at runtime.

2.2 Installing the driver in the system

The script *install_hpcap.bash* only loads the driver in the kernel, but does not actually install it. If you want to use HPCAP as a standard driver, you should run `sudo make install`. You can load the driver with `modprobe hpcap/hpcapvf` depending on which driver you want, or add it to the list of modules loaded at boot (*/etc/modules*). Loading the module with modprobe or at boot time will automatically bring up the interfaces, but will not launch the monitors automatically.

In order to launch the monitors, you can run the command `launch-hpcap-monitors`.

The installation copies the driver binaries to the */lib/modules* folder, the samples and some scripts to */usr/bin* so they're accessible from anywhere, and the libraries and header files in */usr/lib* and */usr/include/hpcap* to build client applications.

2.3 HPCAP folder contents

The first step you must follow to use HPCAP in your system is obtain the HPCAP folder containing all the files related to the corresponding release. Inside this folder you will find the following subdirectories:

- *bin*: Binary files, folder autogenerated by the *Makefile*. Inside this folder there are two subfolders, one for each build configuration *debug* and *release*. You should normally use the binaries in the *release* folder unless you want to debug a specific problem.
- *data*: The monitor log directory. Each HPCAP interface has a line for every containing the current timestamp, captured bytes, captured frames, missed bytes and missed frames.
- *doc*: Documentation files
- *driver*: Driver source code files.
- *include*: Common files (headers) for both driver and user level apps.

- *obj*: Intermediate build files, folder autogenerated by the *Makefile*.
- *srclib*: Source files for the library used by user level level apps.
- *samples*: Several sample applications built for the HPCAP driver, including *raw2pcap* and *hpcapdd*.
- *install_hpcap.bash*: Script that loads HPCAP in the system.
- *params.cfg*: Driver configuration.
- *scripts*: Scripts used to monitor and configure hpcap driver

2.4 Testing the installation

For testing whether the installation is correct or not, we can use the data folder. Here, the monitor programs keep a file for each capture interface, adding a new line every defined time (a second by default) with the captured traffic amount in that period. The *hpcap-status* tool also shows a brief description of the captured traffic and HPCAP driver status.

It is important to notice that for these tools to work, there must be at least one consumer for the interface, like `hpcapdd`.

Chapter 3

Configuration

As specified in the quick-start guide (section 1), the *scripts/gen-hpcap-config* script can generate a basic configuration based on the information of the system and some sensible defaults that should be enough in most cases. However, when that configuration is not enough or when the performance obtained is not the expected one, you should know which are the configuration parameters of HPCAP and be able to tweak them to fit your needs.

3.1 Configuration parameters

All the scripts used for the installation and management of the HPCAP driver use of the information specified in the *params.cfg* file that is located in the root level of the HPCAP folder, or in */etc/hpcap/* if the driver is installed in the system (see section 2 for details). 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:

- `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. It is not recommended to change this unless you know what changing this value implies.
- `consumers`: For the 40G version, controls the number of consumers for each queue. Consumer threads are allocated sequentally in the CPU space.
- `use_vf`: Can be 1 or 0 depending on whether you want to install the virtual function driver or not.
- `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 section 4.2 for details. See section 3.2 for more details on how the interfaces should be named.

  The interfaces in this options should be included in order and without gaps. That is, if you want to configure interfaces 1 and 3 in HPCAP mode, you should write all the interfaces until the fourth one: `ifs=''xgb0 hpcap1 xgb2 hpcap3''`. If you skip any interface, the installation scripts will not work.
- 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 section 3.2). 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.

    A third mode is available: mode 3 will configure the interface with hugepage-backed buffers (see section 3.4). This mode is only used in the script, and internally mode 2 is used in the driver. The

distinction is made only for automating the buffer mapping process with the script.

- `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 6.
- `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.
- `hugesize<itf_index>`: buffer size for interfaces in mode 3. Will be aligned to hugepage size at runtime. Can be a human-readable size, such as 2GB or 4096MB.

- `monitor_enabled`: Can be 1 or 0. If 1 (enabled), the *install_hpcap.bash* script will launch the monitors for the HPCAP interfaces.
- `monitor_basedir`: Where to save the monitoring logs when the driver is installed in the system. The logs will be saved in *HPCAP/data* if the driver is not installed but is running from the source folder.
- `monitor_interval`: Sampling interval, in seconds, for the interface monitors.
- `monitor_core`: Core in which the monitors will run. Useful when the scripts interfer with the kernel or capture threads.

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.

3.2 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_identifier><interface_index>`, where:

- `<mode_identifier>`: 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`.

The number assigned to each interface is fixed across reboots, as it is based on the order assigned by the PCI bus and function index of each port.

3.3 Multicore architectures

Given the performance requirements of HPCAP, a careful configuration in multicore systems is needed. HPCAP launches a polling thread for each interface. These threads should be placed in the same NUMA node as the physical card for optimal performance, using the `core` parameter of each interface (see section 3.1).

In order to get the NUMA node of the NIC, you can either use *lstopo* from the *hwloc* package or read the file *sys/bus/pci/devices/0000:[PCI ID]/numa_node*, where *PCI ID* is the ID of the NIC that you can extract from the output of *lspci*. A sample for both options is provided below.

```
$ lstopo
Machine (128GB)
  NUMANode L#0 (P#0 64GB)
```

```
    ...
    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" <-- NIC in node 0
        PCI 8086:10fb
          Net L#3 "xgb1"
    ...

$ 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 3.1: Obtaining NUMA information.

The capture applications should be placed in cores different than the used for the HPCAP threads. Based on [1], it seems that the optimal configuration is to put the capture applications in a different NUMA node from the HPCAP cores. However, this depends on the specific system you are using, so you should test with different configurations if you are not getting enough performance (that is, HPCAP is losing frames).

You should also ensure that the kernel does not use the cores assigned to the HPCAP threads and capture applications to schedule other processes. Knowing the list of cores to be assigned, you must isolate them using the boot parameters of the system. 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.

3.4 Using hugepages

The HPCAP driver can use hugepage-backed buffers to increase the buffer size and access speed. The install script manages this automatically when interfaces are set in mode 3 (see section 3.1). However, it expects the *hugetlbfs* to be mounted in */mnt/hugetlb*. The script *hugepages_mount* can be used to mount this filesystem automatically.

In order to use hugepages, your system must be configured properly at boot time. That means that you should add the corresponding parameters to the *linux* command in your *grub.cfg* file. These parameters are *default_hugepagesz*, *hugepagesz* and *hugepages*, as documented in the kernel documentation. For example, the parameters `default_hugepagesz=1G hugepagesz=1G hugepages=8` would enable hugepages of 1 GB, and then would allocate 8 hugepages of 1GB at boot time. In the end, your *linux* boot command could look like this:

```
linux /boot/vmlinuz-3.8.0-29-generic root=UUID=b2be13dc-7a60-4928-b61a-cc3ec95044a0 iommu=pt
intel_iommu=on isolcpus=0,1,2,3,4,5 ro default_hugepagesz=1G hugepagesz=1G hugepages=8
```

If you want to make the hugepage configuration persistent even when GRUB is updated, you should add the parameters to the */etc/default/grub* file.

You can check if the hugepages were assigned correctly by running `grep ''Huge'' /proc/meminfo`. More information regarding the inner workings of hugepages can be found in the developer guide of HPCAP.

In order to use hugepages, you have to set the interface in mode 3 and assign it a size with the `hugesizeX` parameter (see section 3.1 for details on the configuration options).

3.4.1 Memory limits

Linux systems usually have a limitation of the amount of memory that can be allocated by a single process. This can cause errors when trying to allocate big buffers backed by hugepages. The first limit to check should be the user limits (*ulimit -a*) enforced by the system.

If the memory is unlimited for the user, then the kernel could be limiting the amount of shared memory and shared memory segments. They can be checked by running the commands `sysctl kernel.shmmax` and `sysctl kernel.shmmni`. The most important is *shmmax* as it limits the amount of shared memory of a process, and has a default value of 4 GB (4294967295 bytes). These limits can be changed by running the commands:

```
sudo sysctl -w kernel.shmmax=[newvalue]
sudo sysctl -w kernel.shmmni=[newvalue]
```

For the memory segments limits, usually doubling the default value of 4096 is enough.

Chapter 4

Usage

4.1 Traffic capture

The capture process can be done as explained in section 1.3 or using other applications that call the HPCAP library. HPCAP provides two simple applications that capture and store traffic.

4.1.1 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 section A.1).

`hpcapdd` has been programmed so it performs an orderly close when receiving a `SIGINT` signal, so it must be ended with `kill -s SIGINT ...` or `killall -s SIGINT ...`.

4.1.2 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 decreases 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 A.1).

As with `hpcapdd`, `hpcapdd_p` has been programmed so it performs an orderly close when receiving a `SIGINT` signal, so it must be ended with `kill -s SIGINT ...` or `killall -s SIGINT ...`.

4.1.3 Reading the capture files

As explained in section A.1, HPCAP generates RAW files. These can be converted to PCAP using the program *raw2pcap*[1].

To check the integrity of the captured files, the program *checkraw* can also be used: it receives as argument the directory where all the files are stored, and then goes on checking all of the frames and that they are saved in the correct format. Optionally, the program can receive a minimum timestamp as a second argument. If it encounters file with an older timestamp, it will ignore them.

4.2 Interface monitoring

HPCAP provides monitoring scripts that log the performance of the capture for each interface. The data generated by those monitoring scripts can be found in the `data` subfolder or, when the driver is installed in the system, in the directory specified in the *params.cfg* file. 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. The fields appearing in each of the *hpcapX* files are:

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

[1] Found with the rest of the applications in the *bin/release* folder after compilation.

Additionally, in the log directory you will find files named *iface-monitor.log* and *iface-monitor.pid* for each interface monitored. These store the output of the monitor (useful to see if any errors have been happening) and the PID of the process, that can be used to terminate it automatically.

To launch and stop the monitors, you can use the *launch-hpcap-monitors* and *stop-hpcap-monitors* in the *scripts* folder or in your PATH if you have installed the driver.

For easy reading of these logs, the script *scripts/hpcap-status* is provided: it will continously print in an easily readable form the status of the drivers: link state, hugepage allocation stats, reception rate and loss stats.

4.2.1 Parsing the monitor log

The script *parse_hpcap_log* has been created to help in the analysis of the HPCAP monitor logs. This script receives two arguments: the log file itself and, optionally, the monitoring interval in seconds (by default it's set to 1).

It then reads the log file, extracting the relevant entries (it does so by stripping entries with zero frames received and lost from the beginning and the end of the file) and parsing relevant details, such as test begin/end date, average, maximum and minimum transfer speeds and lost frames. It also outputs a simple timeline made by grouping the entries depending on whether they had losses or not. It will print something similar to this:

```
At Wed Sep 17 21:53:18 CEST 2014 (ts 1410983598, elapsed 0 secs) all ok (1 secs). Received 755952
    frames, 4256.85 Mbits, 4.157 Gbps.
At Wed Sep 17 21:53:20 CEST 2014 (ts 1410983600, elapsed 1 secs) losses (5 secs). Lost 11083 frames.
     Received 8468847 frames, 48153.52 Mbits, 9.404 Gbps.
At Wed Sep 17 21:53:25 CEST 2014 (ts 1410983605, elapsed 6 secs) all ok (1 secs). Received 1702646
    frames, 9675.18 Mbits, 9.448 Gbps.
At Wed Sep 17 21:53:26 CEST 2014 (ts 1410983606, elapsed 7 secs) losses (4 secs). Lost 16807 frames.
     Received 6617632 frames, 37389.83 Mbits, 9.128 Gbps.
At Wed Sep 17 21:53:30 CEST 2014 (ts 1410983610, elapsed 11 secs) all ok (1 secs). Received 1690432
    frames, 9679.26 Mbits, 9.452 Gbps.
```

This allows to see easily in which intervals the driver performed with zero losses and at which rates, and also to see where and how many frames were lost, and during how much time.

Finally, if the system can output *gnuplot* graphs, it will generate a plot of the monitored speed and frame losses.

4.2.2 Monitoring CPU usage

Apart from the monitring scripts, the script *monitorCPUs.bash* can monitor the CPU usage of the system and related applications. The log, *cpus*, is written in the same directory as the *hpcapX* with the following fields:

```
<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) >
```

4.3 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.

4.4 Collecting diagnostic data

If a failure occurs, the file *scripts/diagnostics* can be used to collect information to help the developer of the driver to diagnose the issue. See the developer guide for details on the information collected.

Chapter 5

API

This section is intended for developers of programs that interact with HPCAP, reading from its buffer. There are two ways to read from the HPCAP buffer, one easier and quick, and the other one more complex but with more possibilities.

5.1 Step 0: installing the libraries

Regardless of the used library, it is recommended to install the libraries to the system to avoid problems with relative routes and similar issues. If you are developing your code in the same machine on which HPCAP will be running and capturing traffic, you can just install the driver with `sudo make install`. This will install the driver (see section 2) and copy the libraries and headers to the respective system locations */usr/lib* and */usr/include*. In this way, you will be able to use the HPCAP libraries just as any other system library.

However, you might be developing your code in one machine different from the one where the driver is installed. For this situation, you can install only the libraries and headers with `sudo make install-libs`: you will be able to compile your program and have autocompletion if your editor provides it, but without installing unnecessary drivers and configurations to your machine.

5.2 libmgmon: Easy access to HPCAP packets

The mgmon library gives easy access to the HPCAP buffer, either packet-by-packet or grouping by flows, using a similar call that what . It is best shown with an example:

```c
#include <stdlib.h>
#include <stdio.h>
#include <libmgmon.h>

/**
 * A structure for easy parsing of Ethernet frames
 * Source: http://lxr.free-electrons.com/source/include/uapi/linux/if_ether.h#L139
 */
struct ethhdr {
    unsigned char   h_dest[ETH_ALEN];       /* destination eth addr */
    unsigned char   h_source[ETH_ALEN];     /* source ether addr    */
    __be16          h_proto;                /* packet type ID field */
} __attribute__((packed));

void handle_packet(uint8_t* payload, struct pcap_pkthdr* header, void* arg) {
    struct my_struct data = (struct my_struct*) arg; // Use the arg pointer to store any
    ↪  data you need to pass down to the handler

    // header->ts contains the timestamp of the packet (struct timeval)
    // header->caplen contains the bytes that were captured for this packet. This is the
    ↪  length of the payload argument.
    // header->len is the length of the packet as was seen on capture. Note that it may not
    ↪  be the same as caplen.

    if (header->caplen < size(struct ethhdr)) {
        // Important: always check caplen (not len) to ensure that you have enough data.
```

```
        printf("Not enough length to parse Ethernet\n");
        return;
    }

    // Example: parse ethernet header.
    struct ethhdr* eth = (struct ethhdr*) payload;

    // ...
}

int main(int argc, char** argv) {
    struct my_struct data;

    // ...
    int cpu = 1; // Choose the CPU carefully (see section on multicore archs) for best
    ↪ performance
    int iface = 0; // The interface to listen on (e.g., hpcap0 is iface 0)
    int qindex = 0; // Queue is usually 0.

    // Start the listening loop
    mgmon_packet_online_loop(cpu, iface, qindex, handle_packet, &data);
}
```

You can stop the receive loop by passing a SIGINT (Ctrl-C) signal to the process.

5.3 libhpcap: Raw access to HPCAP

The libhpcap library provides raw access to the HPCAP structures, and is the base for the other libraries. You should probably not use this library unless you know why you want it and you how HPCAP works and interacts with your program.

Although all the functions are documented in *hpcap.h*, you may want to generate automatic documentation with Doxygen. To do this, run `make doxydoc` and open the file *doc/html/index.html* with your favourite browser.

5.4 Notes for the HPCAP API user

When developing for the HPCAP API, you must have some details in mind:

- HPCAP follows the slowest listener. If your program is too slow, the buffer will fill up and HPCAP will start dropping packets for all applications connected to HPCAP. Keep this in mind when deploying multiple applications reading from the same interface: low performance spikes or hangs in one program will cause loss of packets on *all* applications. You can see if this is happening by reading the output of *hpcap-status* (section 1.4).
- Core assignment is important. Read the section 3.3 on multicore architectures to understand how to assign HPCAP and application cores for maximum performance.

Chapter 6

Frequently asked questions (or maybe not so frequent)

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, 3.11.0, 3.13.0, 3.15.0, 3.19.0.
- **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 `hpcap` 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 section 3.1).

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.

I am not obtaining the expected network capture performance, what is happening The most probable cause is a misconfiguration of the core assignments. See section 3.3 for details, but you should make sure that you have properly set the processor affinity for your storage programs (`dd`, `hpcapdd`) via the `taskset` command) and that the assigned processor is not used by any kernel capture thread.

You can also check the script *hpcap-status* or the counter `rx_hpcap_client_lost_frames` in *ethtool*[1] to check if the losses are due to slow clients or if it's the driver the one losing them.

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 to change 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

[1] Run `ethtool -S hpcapX` to see the counters of the interface.

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 `HPCAP/include/hpcap.h` file:

```
#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 recompile the driver and any of the sample applications based on HPCAP used.

How can I know which driver version am I running? The build system automatically generates the build information and includes it in the driver file. To get it, run `modinfo driver -F version`, where `driver` is either the path to the driver *.ko* file or the name of the driver (*hpcap* or *hpcapvf*) if it has been installed to the system. The output will be something similar to this:

```
> modinfo hpcap -F version
HPCAP v4.1.0 built Sat Aug 22 16:18:05 CEST 2015 IXGBE 3.7.17-NAPI
```

The *v4.1.0* indicates the version of HPCAP. It's automatically generated by the build script based on the output of *git describe*, so if you are not using a stable release you can see something along the lines of *v4.1.0-7-g1237abc*: it shows that the driver has beeen build from the commit with abbreviated hash *1237abc*, 7 commits after the 4.1.0 version. It

The last part of the version string refers to the base IXGBE version that was used to create the HPCAP driver.

The driver installation fails with "Cannot allocate memory" Check the kernel log, further instructions and details should be there. For example, it could be related to the configuration of the pages for each adapter: you should check that there are no more pages assigned that pages are available. The available page count depends on the `HPCAP_BUF_SIZE` macro and is printed in the kernel log when the page configuration is incorrect.

The driver installation fails with "Unknown symbol in module" Check the kernel log. If it says something similar to *hpcap: Unknown symbol vxlan_get_rx_port (err 0)* try running `modprobe vxlan` before installing.

6.1 Bugs, system errors and known issues

This section shows the bugs that have been found during the development and use of the driver, hoping that it gives the reader some ideas when the driver and/or the kernel have turned crazy and decided not to work.

6.1.1 Driver/system corruption after system crash

Status: fixed (*commit: 06b9ec8: insmod: Blacklist hpcap and ixgbe to avoid loading them at . . .*)

Affected versions: 4.2.0, 4.2.1

During the testing of HPCAP 4.2.1, the system crashed because of unknown issues while HPCAP was receiving traffic. The driver was installed to the system. After a reboot, something was corrupted: both *ixgbe* and *hpcap* were loaded at boot (even though none of them was configured to do so) and could not be removed. Trying to install the module again resulted in either errors or *modprobe/insmod* hanging.

It seems that, after a crash, the Linux kernel tries to reload the modules in some weird fashion, reloading both *ixgbe* and *hpcap* and causing some weird conflicts, including errors in the Intel Direct Cache Access subsystem (in some instances, the driver crashed on install with the stack trace pointing to `dca_register_notify`).

The fix was to blacklist both drivers in the *modprobe* configuration that was already being generated on install. When the error happened, the system could be restored running `depmod -a && update-initramfs -u` with *root* permissions, or reinstalling the kernel.

6.1.2 Segmentation fault on client applications

Status: fixed (*commit: 9f21f1f: Fix concurrency issue when adding listeners*)

Affected versions: 4.1.0 and previous

Sometimes, when multiple clients open the same HPCAP handle at the same time (for example, *monitor_flujos* opens two handles on the same file), a race condition may occur and the listener identifications will not be registered correctly. This causes messages in the kernel log warning about some "listener not found" and possibly a segmentation fault when issuing `ioctl` calls. The solution was to properly protect critical sections of the code that can be read and written concurrently.

6.1.3 modprobe/insmod: Cannot allocate memory

Status: not fixed

Affected versions: All

There are several possible causes. If the error is raised by the HPCAP driver, you should see a line in the kernel log explaining what happened and how to correct the error[2].

However, if the kernel log doesn't show any message from *hpcap*, it may be an issue with available memory in the kernel. The Linux kernel has a certain memory space assigned to load modules[3], usually about 1.5GB. It's more than enough for regular uses of modules, but not in the case of HPCAP. HPCAP allocates a static buffer of 1GB that will be shared between the different queues. This buffer is placed on that module mapping space[4] so, if several modules have already reserved more than 500 MB, the kernel may refuse to load our module.

The workaround for this is to reduce the buffer size (modify the macro `HPCAP_BUF_SIZE` in the *include/hpcap.h* file) and, if necessary, to use hugepages (see section 3.4) to get buffers of the necessary size. I haven't found any way to see the memory usage of each driver nor how to see how much of that module mapping space is already allocated.

6.1.4 Interfaces renamed automatically by udev

Status: fixed (*commit: r715: insmod: workaround for udev renaming rules, r807*)

Affected versions: 4.2.2 and previous

In some systems, udev has some "smart" renaming rules that theoretically gives a predictable naming to the network interfaces. That interferes heavily with HPCAP. The workaround consists in a small check in the `interface_up_hpcap` bash function, that will check in the kernel log for *udev* renamings and will undo them.

An extra problem in some CentOS installations was that *udev* did not even print in the kernel log these renames. This required an extra fix (revision 807 in *mellanox* branch, merged into trunk in rev. 838), that consisted of a naming check (the actual function is called `hpcap_check_naming`) that is called before launching the poll threads. This functions compares the current name of the interface to the expected one (*hpcapX*, where *X* is the adapter index): if they are different, it prints a message to the kernel log that will be picked up by the installation script so it can rename the interfaces.

6.1.5 Corrupted capture files

Status: fixed (*commit: r842, r869*)

Affected versions: 4.2.3 and previous

A bug was discovered where the captures were being corrupted if another program had been listening previously. For example, if a first instance of *hpcapdd* was launched and then closed, the resulting capture files were all correct. However, if a second instance was launched without reinstalling the drivers, all capture files would be corrupted.

The actual issue had to to with how the clients (listeners) acquire their reading offsets. A first listener would start reading from offset 0, transferring blocks of a fixed size. When the client closed, HPCAP saved the last reading offset, which would not necessarily point to the beginning of a frame.

Thus, when another new client tried to read from the HPCAP buffer, its reading offset was the last reading offset of the other client. This new client would begin saving the capture at the middle of a frame, so applications such as *raw2pcap* or *detectpro*, expecting to read first a correct packet header, would crash and/or output completely wrong results.

[2] If the message appears but does not tell how to fix the error or does not explain what's happening, tell the maintainer to change that error message.

[3] See the kernel documentation about the matter.

[4] It may be possible that I'm wrong here about this.

A first solution was to assign as read offset the last writing offset of the HPCAP producer. New frames are written starting from that offset, so new listeners would start reading frames with the correct headers. However, this caused another extra problem: misaligned access to the buffer that could hurt performance. A fix for this issue is simply resetting the read/write offsets when there are no listeners: if there are no client applications, HPCAP will forget the last read/write offsets and will start writing frames from offset 0, thus avoiding misaligned access for new clients.

A secondary bug appeared later, introduced by the first fix (r842): in some cases the global write offset would be advanced before the read offset of a new listener was configured. In these cases, the read offset would again be greater than 0 and would cause errors when writing the captures. This bug was fized in r869 by setting the read/write offsets in the intialization of the listener.

6.1.6 Connection failure / segfault on xgb interfaces

Status: fixed (*commit: r826: ixgbe: Fix bug with ixgbe interfaces where next_to_use wasn't ...*)

Affected versions: 4.2.3. Possibly previous versions, not checked

Interfaces configured in the *xgb* mode were failing and/or crashing the system completely when they received data.

The cause was that the `rx_ring->next_to_use` pointer was not updated correctly. It should be updated in `ixgbe_release_rx_desc` to have the same value as the ring's tail pointer of the NIC, but the corresponding line was missing. This caused incoherencies between the ring status in software and in hardware, which in turn caused missing frames (thus the failing connections) and even crashes when the software tried to access to unallocated descriptors.

6.1.7 Connection reset after TX hang

Status: fixed (*commit: r875, r876: ixgbe: Disable transmission and forced resets on TX hangs*)

Affected versions: 4.2.3 and previous

If an application in the system tried to send frames through interfaces configured in HPCAP mode, it could trigger a reset in the interface that would lead to corrupted captures.

The bug was that the *ixgbe* driver would store the packets to send in a queue which was not flushed. Then, a watchdog (either in the driver or in the kernel) would notice[5] and reset the adapter. This would in turn reset the poll thread and corrupt the capture.

The solution was to discard frames to transmit in HPCAP adapters, and to disable the code that resets the adapter when using HPCAP work modes.

6.1.8 Loss of all frames

Status: fixed (*commit: r963: hpcap: Fix bug where the driver was losing all frames*)

A misterious bug where at random times caused the driver to lose all frames. The cause was the padding check. When receiving a packet, if the space left in the capture file was exactly 2 RAW headers plus the incoming packet size, no padding would be written. The space left for the next frame was exactly the size of the RAW header, so the padlen, calculated as the remaining space in file *minus the size of a RAW header* would be zero and no padding would be inserted. The file size would be also incremented, surpassing the value of `HPCAP_FILESIZE` and either corrupting the memory space when writing with erroneous lengths or avoiding the reception of any packets as the driver would not have space to insert the padding with that size.

6.1.9 IOCK and MNG_VETO bit enabled - capture thread stops

Status: not fixed

Affected versions: pre-5.0.0, at least

In a certain installation, the driver would stop capturing after the system received a non-maskable interrupt about an IOCK error. There is not much documentation about this error, but it seems to be related with the hardware.

The kernel log also showed a message from the base *ixgbe* driver about a `MNG_VETO` bit enabled. Reading the NIC datasheet [2], it seems to be a bit set up by the hardware when it is in low-power state that forbids link changes, so downed links would not be restored.

[5] The exact message was *initiating reset due to lost link with pending Tx work.*

As of 27/4/2016, this bug seems to be caused by the hardware.

6.1.10 Concurrent listeners leads to losses and/or data corruption

Status: fixed (*commit: r959, r955*)

Affected versions: 4.2.3 and previous

Several related errors happened when there was more than one client listening for data in a HPCAP interface. For example, if a listener connected first but did not read anything, a second listener would receive an advanced read offset (the last write from the polling thread). So, even when the two listeners could start reading data from the buffer at the same time they would see different data. This was fixed in revision 959.

Another problem happened when two listeners connected to the driver and then the first one disconnected. The next listener to connect would receive handle ID 2, which was already being used. Data loss and/or corruption of the structures would then occur. This was fixed in revision 955.

6.1.11 Copies from/to user space can fail

Status: fixed (*commit: r956: hpcap: Fix copies from/to the user memory*)

Affected versions: 4.2.3 and previous

In the `ioctl` calls, the copies to and from user space memory were not being done with `copy_to/from_user` and, in strange instances where that memory was invalid, that could cause a segfault within the driver. With the fix, no invalid acceses are done and the correct error message is returned to the caller application.

6.1.12 Corrupted capture files after restarting a client

Status: fixed (*commit: r893: hpcap: Reset written buffer size when there're no listeners*)

Affected versions: 4.2.3 and previous

Related to bug 6.1.5, when restarting a listener, the captures would be subtly corrupted as the driver would not place the padding at the end of the file but in the middle instead. This happened because the filesize counter was not being reset if there were no listeners.

6.1.13 Problems when using buffers greater than 2GB

Status: fixed (*commit: r990: hpcap: Fix several problems with buffers greater than 2G*)

Affected versions: 4.2.X

When using hugepage-backed buffers with a size greater than 2GB, several variables would overflow and cause segfaults and memory errors.

There is a secondary problem: when allocating buffers of size 4GB, the function `vm_map_ram` used in the hugepage code will segfault due to a bug in the Linux kernel code. See the linux-mm mailing list for more information. The patch was included in kernel version 4.7.

6.1.14 Frames with VLAN tags disappeared

Status: fixed (*commit: r1025: ixgbe: Fix VLAN stripping*)

Affected versions: pre-5.0.0

During the update of the ixgbe base driver from 3.7.17 to 4.1.2, the VLAN stripping features were not correctly deactivated. This caused the card to strip the VLAN tags in hardware. The solution was to disable all hardware features in the correct location in the code.

6.1.15 RX Errors with no cause

Status: not fixed

Affected versions: All

In some environments, the *ethtool* counters for `rx_errors` increment, but without CRC or length errors. These may be caused by corrupted non-Ethernet packets, such as malformed Cisco Spanning Tree Protocol

(STP) frames. The solution was to make the NIC stop counting these frames as errors, disabling a certain register. This required to add the line

```
hlreg0 &= ~IXGBE_HLREG0_RXLNGTHERREN
```

Appendix A

Configuration example

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

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

####################
# Use virtual function driver
use_vf=0;
####################

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

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

####################
# Interface list
#  E.g.:
#    ifs=hpcap0 xgb1 hpcap2 ...;
ifs=hpcap0 xgb1 hpcap2 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]")
#         3 = high performance RX with hugepage-backed buffer.
# 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=3;
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=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=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 captured 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
# 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=1;
pages1=1;
pages2=131072;
pages3=131072;
```

```
# Note: We don't assign more than 1 page to the interface 0 as it will use hugepages.
####################


####################
# Enable the monitors to be launched automatically.
monitor_enabled=1;
####################

####################
# Monitor script sampling interval (seconds)
#    must be >0
monitor_interval=1;
####################

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


####################
# Where to place the monitor log output.
# Only for installations. HPCAP running from source
# will still save the logs in the data directory.
monitor_basedir=/var/log/hpcap;
####################

####################
# Buffer size for hugepages mappings.
#     Can be human-readable (e.g., 20G, 1024K, 1M)
#     Will be aligned to hugepage size at runtime.
#     Only for interfaces in mode 3.
####################

hugesize0=2G;


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

A.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 [3].

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

A.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 figure A.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.

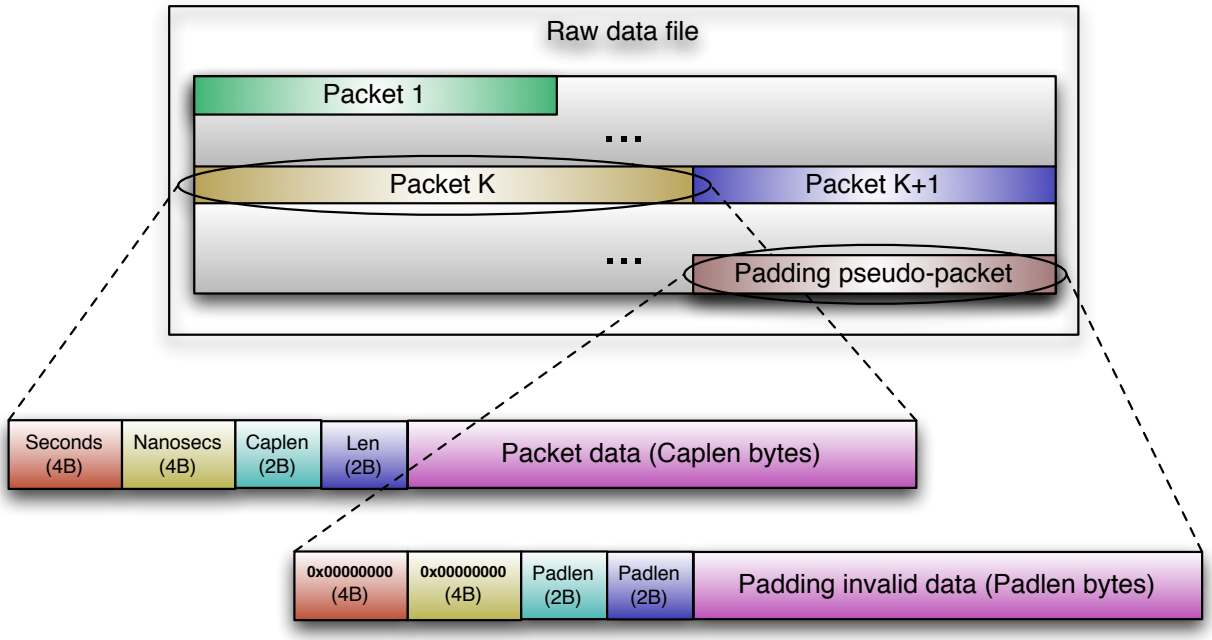

Figure A.1: Raw file format

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.

A.1.2 Example code

The file *raw2pcap.c* in the *samples/raw2pcap* folder is an example of a program that reads a raw file and generates a new pcap file with the contents of the first one.

Bibliography

[1] Víctor Moreno, Pedro M. Santiago del Río, Javier Ramos, David Muelas, José Luis García-Dorado, Francisco J. Gomez-Arribas, and Javier Aracil. Multi-granular, multi-purpose and multi-Gb/s monitoring on off-the-shelf systems. *International Journal of Network Management*, 24(4):221–234, 2014.

[2] Intel. Intel 82599 10 GbE Controller Datasheet. *October*, 2010.

[3] 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.