nmon.net

nCap User's Guide

High Speed Packet Capture and Transmission Library

Version 1.0
December 2004

1. Introduction

nCap is a high speed packet capture and transmission library that turns a commodity PC into an efficient and cheap network measurement box suitable for both packet and active traffic analysis and manipulation. Moreover, nCap opens totally new markets as it enables the creation of efficient application such as traffic balancers or packet filters in a matter of lines of codes. This is because contrary to conventional operating systems where the development of this kind of applications happens into the kernel, nCap is a pure userland library that is based on a special kernel driver. This means that every programmer able to code in C can write such applications without the need to learn the kernel insights.

This manual is divided in two parts:

- nCap installation and configuration.
- nCap SDK.

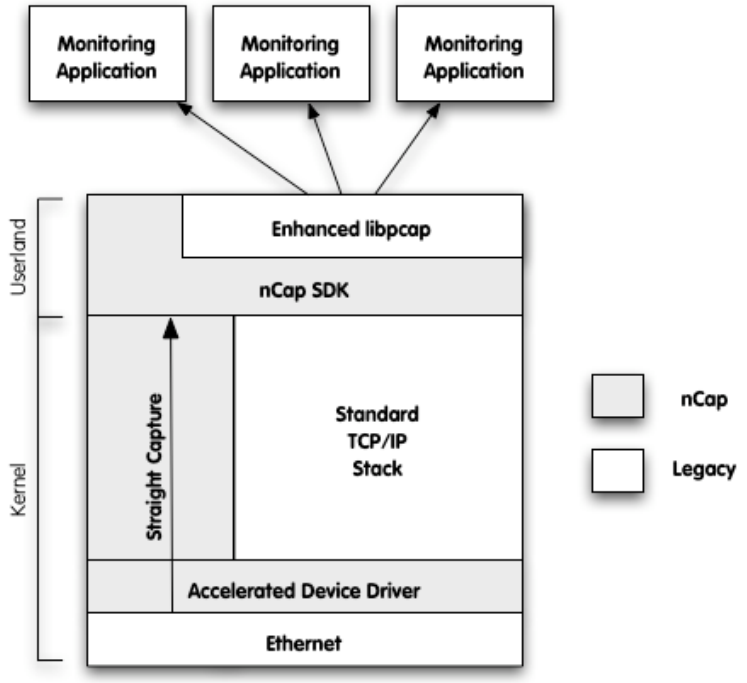
1.1 What's New with nCap?

Release 1.0 (December 2004)

- Initial nCap Release

2. nCap Installation

nCap's architecture is depicted in the figure below.



The main building blocks are:

- The accelerated kernel driver that provides low-level support and ethernet device programming.
- The user space nCap SDK that can be used directly or through an enhanced libpcap that provides transparent nCap-support to legacy pcap-based applications.

IMPORTANT
Currently the accelerated nCap driver is provided only for Intel 1 Gbit (copper, SX and LX) and 10 Gbit Ethernet cards. Therefore you need to have one of these cards in order to take advantage of nCap.

When you download nCap you fetch the following components:
A precompiled Linux kernel with all the available kernel modules.

- The nCap SDK
- The enhanced libpcap based on nCap

2.1 Linux Kernel Installation

The Linux kernel is composed of two parts:

- The vmlinuz-2.X.Y kernel that you need to copy into your /boot directory
- The 2.X.Y/ directory that contains the kernel modules and that needs to be copied into /lib/modules.

Once the kernel is installed you need to modify your boot loader (usually lilo or grub) in order to let your system access the new kernel. Done this, you need to reboot the box and make sure you select the kernel you just installed as default kernel.

Note that:

- the kernel installation requires super user (root) capabilities.
- For some Linux distributions and installation package is provided.

2.2 nCap Device Configuration

When nCap is activated, the available nCap–aware cards are initialized and are ready to use. Applications that want to open the adapters, interact with the cards using a device that should be located under `/dev/ncap/` and named ethX where X is the interface index. In order to build the /dev/ncap tree with the correct files a utility named `mkncapdev` is provided. When run with superuser capabilities (no command line options are required), this utility creates the device tree.

NOTE FOR LINUX 2.6 KERNEL SERIES

On the latest 2.6 and above kernels, on many Linux distributions the /dev tree is created automatically at every boot by an utility named udev. This means that you either need to configure udev or run the `mkncapdev` at every boot.

2.3 nCap and Libpcap Installation

Both nCap and libpcap are distributed in binary format for immediate usage. They come with some include (.h) files and the object libraries: libncap.a and libpcap.a.

The installer copies the above files in the following locations:

- Include files are installed on /usr/local/include
- Library files are installed on /usr/local/lib

IMPORTANT

Legacy pcap-based applications need to be recompiled against the new libpcap and linked with both libncap.a and libpcap.a in order to take advantage of nCap. Do not expect to use nCap without recompiling your existing application.

2.4 nCap License Installation

nCap requires a license for each ethernet card you use. The license code is computed on the MAC address on the card on which nCap is activated. You need to copy the activation code you received for your ethernet card on the /etc/ncap.license file. You can watch into the syslog for nCap license issues.

This is a simple license file that contains the activation code for adapters eth1 and eth2:

> cat /etc/ncap.license

eth2 checksum
D235B2930CF18C3A8A03784E693C1674

eth1 checksum
B1255C1D95275E16252FE6DB145E0DBF

At this point nCap is installed and ready to use. If you plan to use it through the enhanced libpcap or installing some nCap aware applications such as those provide by nmon, you don't have to perform any other installation steps and you can use your applications right away. Instead if you use third party nCap-based applications, please refer to the installation guide of these products for further details.

3. nCap for Application Developers

Conceptually nCap is a simple yet powerful technology that enables developers to create high-speed traffic monitor and manipulation applications in a small amount of time. This is because nCap shields the developer from inner kernel details that are handled by a library and kernel driver. This way developers can dramatically save development time focusing on they application they are developing without paying attention to the way packets are sent and received.

This chapter covers:

- The nCap API.
- Extensions to the libpcap library for supporting legacy applications.
- How to patch the Linux kernel for enabling nCap

3.1 The nCap API

The nCap internal data structures should be hidden to the user who can manipulate packets and devices only by means of the available API defined in the include file ncap-int.h that comes with nCap

3.1.1 Return Codes

By convention, the library returns negative values for errors and exceptions. Non-negative codes indicate success. Please always use the code specified by nCap and not a numeric value as the mapping between code and value is not guaranteed in future library versions.

On success nCap can return the following codes:

Code	Description
NCAP_NO_ERROR	Success

nCap can return the following error codes:

Code	Description
NCAP_ERR_NO_SUCH_DEVICE	The specified device cannot be found on the system. Make sure that the name is not misspelled and that the device is both present (e.g. using ifconfig) and nCap-aware. Valid names are 'eth0'`
NCAP_ERR_MAPPING_FAILED	The kernel driver cannot map to nCap some kernel structures. Look at the syslog for further explanations.
NCAP_ERR_DEVICE_UNAVAILABLE	The device is unavailable (i.e. present but down or in use).
NCAP_ERR_INVALID_LICENSE	The nCap license (see /etc/ncap.license) for the specified device is either not available or wrong.
NCAP_INTERNAL_ERROR	Internal nCap error. This error code should not happen. Please report to nmon about this problem including how to reproduce it.
NCAP_TX_SLOT_NOT_YET_AVAILABLE	The packet cannot be sent as the adapter has no slot available for accommodating the packet.. This error code is returned only for packet transmission.

3.1.2 nCap: Device Initialization

*struct ncap_device_info *init_ncap_device(char *dev_name, int *error);*

This call is used to initialize an nCap device hence obtain a handle of type struct ncap_device_info that can be used in subsequent calls.

Input parameters:

- dev_name
  Symbolic name of the ncap-aware device we're attempting to open (e.g. eth0).

Output parameters:

- error
  A numeric error code if the call fails.

Return value:

- On success a handle is returned, NULL otherwise.

3.1.3 nCap: Device Termination

*void term_ncap_device(struct ncap_device_info *dev);*

This call is used to terminate an ncap device previously open. Note that you must always close a device before leaving an application. If unsure, you can close a device from a signal handler.

Input parameters:

- dev
  The nCap handle that we are attempting to close.

Return value:

- None regardless of the status.

3.1.3 nCap: Send a Packet

*int ncap_send_packet(struct ncap_device_info *dev, uint16_t packet_len, unsigned char *packet);*

This call is responsible for sending packets out of the specified device. Note that as the adapter may be busy with other operations, it is not guaranteed the packet delivery, hence it is necessary to look at the return code without assuming (wrongly) that the specified packet has been transmitted successfully. Note that the packet is sent immediately and not delayed (as it needs to be passed to the kernel, then to the target adapter) as it happens in most operating systems

Input parameters:

- dev
  The nCap handle where we are attempting to send the packet.
- packet_len
  The length (in bytes) of the packet we're attempting to send.
- packet
  A pointer to the raw packet. Note that this must be a complete packet (i.e. from the ethernet address up) and not just the packet payload.

Return value:

- Either NCAP_NO_ERROR on success or NCAP_TX_SLOT_NOT_YET_AVAILABLE if the adapter cannot send the packet as the transmit queue has not empty slot to accommodate the packet. This is a common situation if a fast sender tries to transmit packets on a slow media (e.g. 10 Mbit ethernet). Make sure you always check the return code before assuming the packet has been transmitted successfully. In case there are no slots available you can sleep and retry, but do not expect that nCap retries on your behalf as this is not the specified behavior.

3.1.4 nCap: Check Whether There is an Incoming Packet Available

*int ncap_recv_packet_available(struct ncap_device_info *dev, int wait_for_packet);*

This call is designed to either check whether there is an incoming packet available, or check and wait (no active wait) if not available.

Input parameters:

- dev
  The nCap handle where we perform the check.
- wait_for_packet
  If 0 we simply check the packet availability, otherwise the call is blocked until a packet is available.

Return value:

- 1 if there's a packet available, 0 otherwise.

3.1.5 nCap: Read is an Incoming Packet

```
int ncap_recv_packet(struct ncap_device_info *dev, unsigned short *recv_packet_len,
                     unsigned char* packet_buffer, unsigned short packet_buffer_len,
                     int wait_for_packet);
```

This call returns an incoming packet when available.

Input parameters:

- dev
  The nCap handle where we perform the check.
- packet_buffer
  A memory area allocated by the caller where the incoming packet will be stored.
- packet_buffer_len
  The length of the memory area above. Note that the incoming packet is cut if the incoming packet is too long for the allocated area.
- wait_for_packet
  If 0 we simply check the packet availability, otherwise the call is blocked until a packet is available.

Output parameters:

- recv_packet_len
  The actual size of the incoming packet, from ethernet onwards.

Return value:

- 1 if a packet has been received, 0 otherwise.

3.1.5 nCap: Enable/Disable Interface Promiscuous Mode

*int ncap_set_if_promisc(const char *device, int set_promisc);*

By default nCap does not set the promiscuous mode when and interface is open. If required this call does the job.

Input parameters:

- device
  The device name where we want to set the promiscuous mode.
- set_promisc
  If 1 the promiscuous mode is set; if 0 it is unset.

Return value:

- NCAP_NO_ERROR if succeeded, an error code otherwise.

3.2 The Extended libpcap API

The libpcap that comes with nCap has been transparently extended with nCap support. This means that existing applications can be relinked with this library to take advantage of nCap without changing a single line of code. Moreover, as nCap allows packets to be transmitted, the extended libpcap allows it too.

int pcap_send_packet(pcap_t handle, u_char *packet, u_int plen)*

Input parameters:

- handle
  The pcap handle where we are attempting to send the packet.
- packet
  A pointer to the raw packet. Note that this must be a complete packet (i.e. from the ethernet address up) and not just the packet payload.
- plen
  The length (in bytes) of the packet we're attempting to send.

Return value:

- 1 if the packet has been transmitted, an error code otherwise. Note that contrary to ncap_send_packet, this function does not return until the packet has been sent or an error occurs.

3.3 Patching the Linux Kernel: How to Add nCap to a Vanilla Kernel

If you want to patch a vanilla kernel to make it nCap aware you can do it this way:

- Copy ncap.h into include/linux
- Copy e1000_ncap.[ch] into drivers/net/e1000
- Patch drivers/net/e1000/e1000.h and drivers/net/e1000/e1000_main.c as specified into the patch file.
- Configure the kernel so that you include the driver for the Intel Gigabit card (e1000) with the NAPI option enabled.
- You can now build your kernel (make bzImage modules).