nCap: Wire-speed Packet Capture and Transmission

L. Deri
ntop.org
Pisa
Italy
deri@ntop.org

Abstract

With the increasing network speed, it is no longer possible to capture and transmit network packets at wire-speed using general-purpose operating systems. Many companies tried to tackle this problem by manufacturing costly network adapters able to keep up at high network speeds.
This paper describes a new approach to wire-speed packet capture and transmission named nCap based on commercial network adapters rather than on custom network adapters and software.

Keywords
Packet capture and transmission, high-speed network monitoring, Linux.

1. Introduction

Over the past few years, many people tried to improve the performance of packet capture and transmission on host PC. On the software side, the elimination of kernel livelock while processing interrupts [10] as well the use of device polling [13] has significantly improved packet capture performance and eliminated the risk that the operating system becomes unusable when handling packets at high-rate.
Solved the kernel livelock problem, the research community has put a lot of energy for improving the kernel performance while moving packets from the kernel-space to the user-space where monitoring application are living. The first step has been to completely remove the use of per-packet system calls (usually read() and poll()) in favour of memory-mapped approaches [12]. In this case, incoming packets are copied from the NIC to a kernel buffer that is shared between the kernel and the monitoring application by means of memory map. The author has further improved this technique by defining a new type of network socket named PF_RING [4] that in addition to memory mapping, significantly improves the packet capture performance by dramatically reducing the packet journey from the NIC to user-space.

On the hardware side, most (if not all) gigabit adapters support a technique named interrupt mitigation [7] and prioritization that allows multiple interrupts to be combined in order to reduce the amount of interrupts that the kernel has to handle. In

addition, the latest PCI specification defines a new type of interrupt named memory-based interrupt that allows interrupts to be transparently dispatched without any kernel intervention to specific memory areas (e.g. a device driver), contrary to what happens today where the kernel receives all the interrupts and then dispatches them to the appropriate device driver. Furthermore, the new PCI Express [8] specification is further enhancing the performance and bandwidth provided by the PCI/PCI-X bus while preserving software compatibility so that existing network device drivers can run (mostly) unmodified on the new bus, while taking advantage of a new performance dimension.
While the market is smoothly improving commercial network adapters, some companies [5] [9] and universities [2] designed custom network adapters able to efficiently capture packets at high speeds. They have been designed for maximum performance, so very they sport advanced features such as on-board packet filtering, a coprocessor card for computing network statistics directly on the adapter, and optimized (burst) packet transfer of packets on the PCI bus. The drawback of these solutions is mainly the adapter price that can very well be 10/100 times greater than a commercial network adapter of the same speed, making these solutions suitable only for vertical markets.

2. Redesigning Network Packet Handling

While working at PF_RING, the author realized that:

- Many, if not all, passive packet capture solutions, including PF_RING, were designed only for network monitoring, although with a little effort they could have been targeted for a broader scope.
- The ability to operate at wire-speed at least at 1 Gbit is a requirement in many situations. Although simple "filter packet & count it" is a rather easy problem, wire-speed complex traffic analysis (e.g. using NetFlow/IPFIX) may become an issue even on fast computers equipped with the state of the art network monitoring adapters.
- User-space packet transmission at wire speed is often not considered as important as packet capture, that instead is often conceived as an activity for network applications that live into the kernel of the operating system.
- Active network monitoring tools that need to compute precise measurements (e.g. one-way delay) are still sitting on top of standard kernels that introduce an unpredictable latency when transmitting packets that can lead to wrong measurements. This is also because the research has focused mostly on reducing latency on user-space packet capturing rather than transmission.

Therefore the author decided to tackle the above issues by designing a new solution named nCap that satisfied the following goals:

- Wire-speed packet capture and transmission, at least at 1 Gbit.
- Based on commodity hardware and software (i.e. no costly software/adapters are required).
- Ability to develop nCap-based solutions from user-space (i.e. no need to

hack into the kernel).

The idea was to give people the ability to:

- Create both active and passive network monitoring tools with little effort.
- Leverage the knowledge and expertise needed to implement application such as bridges, routers and traffic balancers as everyone able to code in C can implement them quite easily in user-space instead of kernel-space as it happens today.
- Extreme efficiency, even greater of the same application implemented in kernel-space.

The idea behind nCap is to use the kernel to instrument the network adapter that will then be accessed directly from user-space by means of a device that usually sits in /dev/ncap/<device name>. This means that the kernel holds and controls the adapter as long as the nCap device corresponding to the adapter is not in use, whereas as soon as an application opens the device, the application takes over the control of the adapter and the kernel cannot longer send/receive from it.

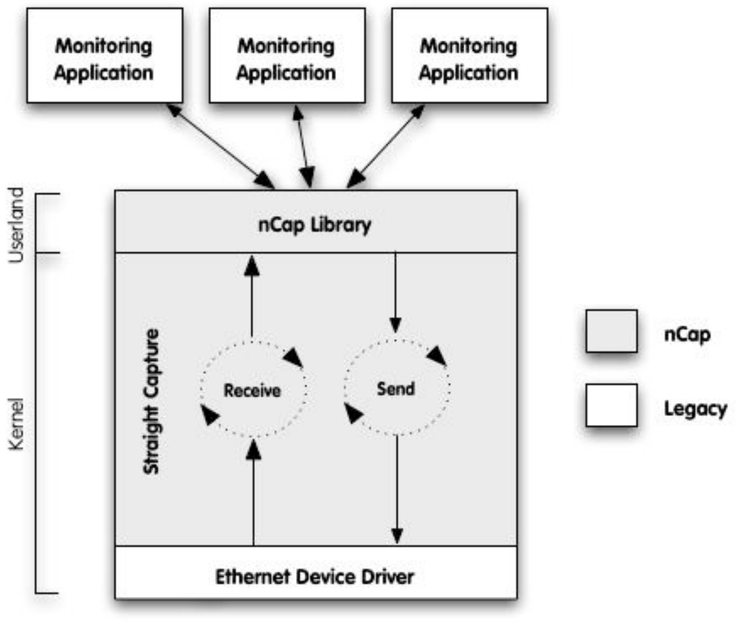

Figure 1: nCap Architecture

nCap is divided in two components: a device driver and a user-space library. The device driver[1] is responsible for controlling the ethernet device and creating two

[1] Currently only the Intel 1 and 10 GE ethernet adapters [6] are supported. Future releases will likely add

circular buffers where incoming and outgoing packets are placed. These buffers are created when the ethernet adapter is initialized. The adapter is instrumented such that the buffers are managed directly by the ethernet adapter itself, via the ethernet controller present on the adapter, without any kernel intervention. This means that as soon as the adapter receives a packet, it copies autonomously the packet into the receive buffer and then alerts the kernel by means of an interrupt (unless interrupts have been disabled). At this point the kernel consumes the packet and updates the receive index buffer.
In the other direction, whenever the kernel wants to send out a packet, it copies it into a free buffer slot and then updates the transmit index buffer. As the buffer indexes are basically a couple of register that sit into the adapter, the adapter knows immediately whenever a packet has been consumed or if there is a packet to transmit without any other delay. This means that the two circular buffers are shared between the adapter and the kernel and that they operate independently in full duplex mode.

The nCap library:

- Enables applications to manipulate the two buffers and their indexes directly from user-space by means of memory mapping without any kernel intervention. This means that nCap creates a straight path from the adapter to the user-space and that the adapter works directly for the application as packets are copied from/to the circular buffers directly by the ethernet controller and not by the kernel.
- Shields the application from knowing the inner details of the device driver or the adapter itself, as it offers a simple API.
- Can sit under the industry-standard libpcap [11] library so that legacy applications can immediately take advantage of nCap without any change.

nCap allows multiple adapters to be open simultaneously by either the same or different applications. The device driver prevents the kernel from manipulating all the open nCap devices. This is necessary for preventing the kernel and application manipulate the buffers of the same interface at the same time, resulting in an inconsistent buffer configuration. This means that when the nCap device is open, the application has full control over the network interface, whereas when the device is not open the kernel controls it as every non-nCap interface.
As said before, when the nCap device is open, the application controls the network interface without any kernel intervention. The previous statement is mostly correct, as there is an exception. In fact when there are no incoming packets to receive, the only thing the adapter could do is to actively loop waiting until a packet arrives. This is obviously a sub-optimal solution as it can waste all the available CPU cycles. For this reason the nCap device implements the poll() system call as follows:

- Interrupts for open nCap devices are disabled.
- If an application calls poll() over an nCap device, the device driver enables the interrupts for the interface and waits for them. As soon as an interrupt

support for additional network adapters.

arrives, the poll() system call returns and the interrupts for the interface are disabled again.

This solution allows applications to avoid wasting CPU cycles while waiting for incoming packets to arrive.

From the nCap architecture just described, it is rather straightforward to understand why this solution is a major step ahead in terms of performance:

- The kernel is completely bypassed during packet capture and transmission.
- The ethernet controller, by means of the shared buffers, brings incoming packets to user-space for immediate consumption.
- The buffers are pre-allocated at startup making un-necessary packet buffer allocation/deallocation whenever a packet is being received/transmitted, as happens with non-nCap aware device drivers.
- Device polling (e.g. on Linux it is called NAPI) is implemented by the kernel, which often can use only the first available CPU. This means that even on a multiprocessor architecture only the first CPU performs device polling. With nCap instead, a multithreaded application can fetch incoming packets fully exploiting multiprocessing as the operating system spawns threads across all the available CPUs.
- Packet transmission is very efficient as it is basically a copy of a packet to the first available buffer slot and buffer index increment. This enables the creation of efficient traffic generators that can sport mostly the same performance of costly ASIC-based traffic generators.
- Since packet transmission starts as soon as the buffer index is incremented, there is very little latency from the time the application decides to transmit the packet and the time the packet is really sent out on the wire. This means that an nCap-based appliance (e.g. a router) is very efficient in terms of packet latency.

3. Inside nCap

The nCap core functions are:

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

Figure 2: Core nCap API Calls

These functions allow packets to be both sent and received though an nCap device handler previously open. The library completely shields the application from low-level details that do not float at user-space. Just to make an example, the following code fragments implements a simple packet balancer based on nCap in a few lines of code.

```
int main(int argc, char* argv[]) {
  /* Open devices */
  while(1) {
    char packet[2048];
    unsigned short packet_len = sizeof(packet);
    int rc;
    struct ncap_device_info *egress;

    if(ncap_recv_packet(in_dev, &packet_len,
                        packet, sizeof(packet), 1)) {
     ncap_send_packet(find_out_dev(packet, packet_len),
                                   packet_len, packet);
    } else {
      printf("recv_packet(): error %d returned\n", rc);
      break;
    }
  }

  /* Close devices */
  return(0);
}
```

Figure 3: Simple nCap-based Traffic Balancer

From the performance point of view, nCap has been designed to be efficient and able to operate at wire speed at least on 1 Gbit networks. In order to evaluate its performance, nCap has been compared against PF_RING. The traffic generator used in the tests for generating packets is home-grown; two ports of a Linksys gigabit switch have been put in loop with a cable, while from a third port a packet of the specified size with broadcast address as destination was injected into the switch. Due to the port loop, the packet runs forever into the switch at maximum speed. The following table compares (see table 6 of [4]) the packet capture performance of nCap and PF_RING solutions in the same test environment.

Packet Size (bytes)	**Linux 2.4.X/RTIRQ NAPI+PF_RING** (Receiver Pentium 4 1.7 GHz, Intel GE 32-bit)		**Linux 2.4.X/nCap** (Receiver Pentium III 550 MHz, Intel GE 32-bit)	
64	550’789 pps	~202 Mbit	561’078 pps	~205 Mbit

Table 1: PF_RING vs. nCap

The test outcome shown that nCap on a Pentium III machine performs roughly as

PF_RING on a Pentium IV[2]. On the same testbed using a 64-bit PCI-X bus and a fast CPU (e.g. Pentium IV HT or Xeon) nCap captures at maximum Gbit Ethernet speed (1.48 Mpps) using a portion of the available CPU cycles, hence leaving spare cycles for packet analysis. It is worth to remark that the entire packet handling in nCap is managed by the ethernet controller and not by the kernel. This means that the nCap performance is not affected by the packet size as it usually happens in other solutions.

4. nCap Evaluation

The idea to implement network applications outside of the kernel is not novel as it already appeared in some operating systems such as the BeOS [1]. What's novel with nCap is:

- Ability to receive/send packets from user-space without any kernel intervention while the application controls the entire packet handling process.
- Minimal packet latency during both capture and transmission.
- Backward compatibility: nCap-enabled interfaces can operate as standard adapters when not accessed through the nCap device.
- Ability to accelerate legacy pcap-based applications by means of the libpcap-over-nCap library.
- Extreme speed regardless of the packet size, as all the traditional software layers such as kernel, packet memory handling, and kernel-to-userspace copy have been completely bypassed.
- Ability to operate at wire-speed using commercial adapters instead of custom interface adapters.

Another advantage of nCap is the ability to develop applications completely in user-space without any need to code into the kernel. This is particularly appealing for education and research as it allows people to prototype applications at user-space that will operate much faster of similar nCap-unaware applications living into the kernel.

5. Work in Progress

Although the current nCap implementation is based on the Intel 1/10 GE driver, due to limited budget it has been tested and evaluated only on 1 GE networks. As 10 GE ethernet is becoming increasingly cheap and spread, it is worth to evaluate nCap also at 10 Gbit speed and compare its performance against custom network adapter adapters.

A limitation of nCap is that when the adapter is open by an application, another application cannot open it at the same time. The author is investigating whether using shared buffer techniques [3] it is possible to remove this limitation without impact on performance.

[2] It is worth to remark that PF_RING performs much better than the standard Linux kernel.

6. Final Remarks

nCap is a solution for wire-speed packet capture and transmission. It has been designed for maximum efficiency and ease of use. Its user-space API makes it easy to use for developing network monitoring applications as well as network appliances including routers and traffic balancers. At least at 1 Gbit, nCap proved to be able to operate at wire-speed on modern PCs while leaving spare CPU cycles for additional activities such as packet analysis. Finally its ability to send and receive packets with very little latency makes nCap suitable for both active and passive network monitoring, as well for students and researchers who want to build powerful traffic management tools without the need to know the secrets of the Linux kernel.

7. Availability

This work is distributed under a dual license: GNU GPL2 for the kernel portion of nCap and BSD for the user-space library and tools. Further information about this work and its availability can be found at the ntop site (http://www.ntop.org/) or contacting its author.

Acknowledgment

The author would like to thank Yuri Francalacci <yuri@ntop.org> and Fulvio Risso <fulvio.risso@polito.it> for the valuable discussions, suggestions and support throughout this research work.

References

[1] The Be Development Team, *Be Developer's Guide*, ISBN 1565922875, 1997.

[2] T. Kratochvíla and others, *Verification of COMBO6 VHDL Design*, CESNET Technical Report 17/2003, 2003.

[3] L. Degioanni and G. Varenni, *Introducing Scalability in Network Measurement: Toward 10 Gbps with Commodity Hardware*, Proceedings of IMC '04, 2004.

[4] L. Deri, *Improving Passive Packet Capture: Beyond Device Polling*, Proceedings of SANE 2004, 2004.

[5] *The DAG Project*, Univ. of Waikato, http://dag.cs.waikato.ac.nz/.

[6] Intel Inc., *Gigabit Ethernet Technology and Solutions*, White Paper, 2001.

[7] I. Kim, J. Moon, and H. Y. Yeom, *Timer-Based Interrupt Mitigation for High Performance Packet Processing*, Proceedings of 5th International Conference on High-Performance Computing in the Asia-Pacific Region, 2001.

[8] PCI-SIG*, PCI Express: Performance Scalability for the Next Decade*, http://www.pcisig.com/specifications/pciexpress, 2005.

[9] Napatech A/S, *The Napatech Traffic Analyzer Solution – White Paper*, 2005.

[10] J. Mogul and K. Ramakrisnan, *Eliminating Receive Livelock in an Interrupt-Driven Kernel*, Proceedings of 1996 Usenix Technical Conference, 1996.

[11] Lawrence Berkeley National Labs, *libpcap*, Network Research Group, http://www.tcpdump.org/.

[12] P. Wood, *libpcap-mmap*, Los Alamos National Labs, http://public.lanl.gov/cpw/.

[13] L. Rizzo, *Device Polling Support for FreeBSD*, BSDConEurope Conference, 2001.