On Multi–gigabit Packet Capturing with Multi–core Commodity Hardware

Nicola Bonelli, Andrea Di Pietro, Stefano Giordano, and Gregorio Procissi
CNIT and Università di Pisa, Pisa, Italy

Abstract. Nowadays commodity hardware is offering an ever increasing degree of parallelism (CPUs with more and more cores, NICs with parallel queues). However, most of the existing network monitoring software has not yet been designed with high parallelism in mind. Therefore we designed a novel packet capturing engine, named PFQ, that allows efficient capturing and in–kernel aggregation, as well as connection–aware load balancing. Such an engine is based on a novel lockless queue and allows parallel packet capturing to let the user–space application arbitrarily define its degree of parallelism. Therefore, both legacy applications and natively parallel ones can benefit from such a capturing engine. In addition, PFQ outperforms its competitors both in terms of captured packets and CPU consumption.

1 Introduction and Motivation

Monitoring high performance links on a current network is definitely a challenging task: on one hand the data rate, which is becoming increasingly high, calls for hardware acceleration of the fast data path, while, on the other hand, the complexity of the analysis to be carried out and the need to have it updated according to the emerging applications and threats requires a flexibility and modularity that only software can provide. However, the evolution of commodity hardware is pushing parallelism forward as the key factor that may allow software to attain hardware–class performance while still retaining its advantages. On one side, commodity CPUs are providing more and more cores, while on the other modern NICs are supporting multiple hardware queues that allow cores to fetch packets concurrently (in particular, this technology is known as Receive Side Scaling, henceforward RSS). Unfortunately, current network monitoring and security software is not yet able to completely leverage the potential which is brought on by the hardware evolution: even if progress is actually being made (multiple queue support has been included in the latest releases of the Linux kernel), much of current monitoring software has been designed in the pre–multicore era. The aim of our work is to make the full power of parallel CPUs available to both traditional and natively parallel application, through efficient and configurable in–kernel packet flow aggregation. Therefore, we designed a novel packet capturing engine, named PFQ, that allows to parallelize the packet capturing process in the kernel and, at the same time, to split and balance the captured packets across a user–defined set of capturing sockets. This
On Multi–gigabit Packet Capturing with Multi–core Commodity Hardware

way, the application writer can arbitrarily choose its level of parallelism with
PFQ, hiding within the kernel the full parallelism of the system. In particular,
an application can either use a single capturing socket (as in the case of legacy
applications) or have PFQ balance incoming frames across a configurable set of
collection points (sockets) or even use a completely parallel setup, where packets
follow parallel paths from the device driver up to the application. In all of those
cases, PFQ yields better performance than its competitors, while burning a lower
amount of CPU cycles. Differently from many existing works for accelerating
software packet processing, PFQ does not require driver modification (although
a minimal few–lines patch in the driver can further improve performance). Scal-
ability can be achieved through batch processing (which, in turn, leverages the
hierarchical cache structure of modern CPUs) and through lockless techniques,
which allow multiple threads to update the same state with no locking and min-
imal overhead. In particular, we designed a novel double buffer multi–producer
single–consumer lockless queue which allows high scalability. PFQ is open–source
software released under GPL license and can be freely downloaded at [1]. The
package consists of a Linux kernel module and of a C++ user–space library.
The rest of the paper is organized as follows: section 2 summarizes the state of
the art in the topic of packet capturing solutions, while section 3 describes the
architecture of our packet capturing engine. Section 4 shows the results of our
measurement campaign and the Conclusions section follows.

2 State–of–the–Art in Packet Capturing

Several solutions have been proposed to speed up the packet capturing capa-
bilities of commodity PCs. nCap [2] uses memory mapping to directly expose
to the application the memory areas where the NIC copies incoming frames.
The same approach is adopted by Netmap [3], a BSD based project which in-
tegrates in the same interface a number of modified drivers mapping the NIC
transmit and receive buffers directly into user space. Also PF_RING [4] uses a
memory mapped ring to export packets to user space processes: such a ring can
be filled by a regular sniffer (thus using the standard linux capturing mecha-
nisms) or by specially modified drivers, which skip the default kernel processing
chain. Those can be both drivers with minimal patches (aware drivers) or heav-
ily modified ones. Memory mapping has also been adopted by the well-known
PCAP capturing libraries [5]. In the past years, the capturing stack of Free-BSD
has been enhanced by a double–buffer mechanism, where packets are written
into a memory–mapped buffer which is first filled within the kernel and then
switched over to the application for reading. This is different from PF_RING,
where applications and kernel work on the same ring concurrently. Although our
proposed architecture also adopts a double buffer solution, it brings it further
by introducing other optimizations (like batch processing) and by explicitly tai-
loring it to a multi–core scenario. Many works (most of them on software based
routers) have obtained good results in accelerating software packet processing
by extensively patching the device drivers. TNAPI [6] effectively addressed the