Wire-Speed Hardware-Assisted Traffic Filtering with Mainstream Network Adapters

Luca Deri [1], Joseph Gasparakis [2], Peter Waskiewicz Jr [3], Francesco Fusco [4 5]

[1] ntop, Pisa, Italy
[2] Intel Corporation, Embedded and Communications Group, Shannon, Ireland
[3] Intel Corporation, LAN Access Division, Hillsboro, OR, USA
[4] IBM Research - Zurich, Rüschlikon, Switzerland
[5] ETH Zurich, Switzerland

deri@ntop.org, joseph.gasparakis@intel.com,
peter.p.waskiewicz.jr@intel.com, ffu@zurich.ibm.com

Abstract. Modern computer architectures are founded on multi-core processors. In order to efficiently process network traffic, it is necessary to dynamically split high-speed packet streams across cores based on the monitoring goal. Most network adapters are multi-core aware but offer limited facilities for assigning packets to processor cores.
In this paper we introduce a hybrid traffic analysis framework that leverages flexible packet balancing mechanisms available on recent 10 Gbit commodity network adapters not yet exploited by operating systems. The main contribution of this paper is an open source hardware-assisted software layer for dynamically configuring packet balancing policies in order to fully exploit multi-core systems and enable 10 Gbit wire-speed network traffic analysis.

Keywords: High-speed network traffic monitoring, hardware-assisted dynamic packet filtering, commodity hardware, operating system design.

1 Introduction

The complexity and heterogeneity of monitoring tasks, such as anomaly and intrusion detection, traffic classification and application level analysis [1], gradually caused a shift from dedicated network devices toward hybrid software and hardware architectures which are more flexible and easier to maintain than dedicated monitoring devices [2]. Along with hardware-based solutions [20], researchers have demonstrated that the performance of traffic analysis applications running on commodity hardware can be substantially improved by properly accelerating selected operating system tasks [19, 21, 22]. However, the performance gap between pure software solutions and hardware assisted ones has been significant. Recent advances in off-the-shelf server technologies suggest that the gap can be substantially reduced. In fact, modern servers are based on advanced multi-core processors featuring

integrated memory controllers and high-speed and low latency interconnections. In addition, off-the-shelf network interface cards (NICs) are supporting new advanced features such as message signaled interrupts (MSI-X), multi-queue capabilities and virtualization support, which have been designed to boost the network performance in specific scenarios. The trend is to introduce into NICs the logic for offloading workstations from computationally intensive network operations. With the advent of multi-core processors, balancing the networking workload among cores is necessary in order to increase the networking performance of network services. Therefore, modern interface cards provide multiple independent reception (RX) and transmission (TX) queues and hardware traffic splitting techniques to distribute the traffic among cores.

Unfortunately, traffic monitoring software did not fully benefit from these new breakthrough technologies. The reason is that software layers on top of which network monitoring applications are implemented, such as network device drivers and operating systems, are not designed for exploiting these features for network monitoring purposes.

In this work we present a flexible and extensible framework that simplifies the development of complex and yet efficient traffic analysis applications running on commodity hardware. The main contribution of this work is a novel traffic balancing and filtering networking layer optimized for traffic analysis purposes that fully exploit advanced features implemented by modern off-the-shelf NICs. The framework is characterized by the following properties:

- It provides an API for hardware-assisted traffic filtering and balancing across cores.
- It can be deployed on sub-1000$/port commodity network adapters which are more than an order of magnitude cheaper than dedicated traffic monitoring devices.
- The filtering mechanisms are flexible and able to address common problems monitoring scenarios such as adaptively balancing the incoming traffic among cores or dynamically filtering incoming traffic.
- It can be used as a building block for designing complex yet efficient monitoring applications.
- It is publicly available at no cost under the GNU GPL license.

The rest of the paper is structured as follows. In section 2 we describe how the software framework we designed few years ago could benefit from modern NICs in particular for supporting in hardware those features we previously implemented in software. In section 3 we position the work described in this paper against similar efforts. Section 4 describes the design and implementation of a new software layer that allowed us to offload traffic filtering to modern NICs. Finally section 5 describes some common use cases we used to evaluate the developed solution hence to demonstrate that this work is a major step ahead with respect to existing software-only solutions.

2 Motivation and Scope of Work

The intrinsic dynamism of Internet protocols has increased the demand for flexible monitoring frameworks designed to speed up the development of efficient and cost effective applications capable to analyze modern network protocols. Nowadays, most network monitoring infrastructures are built around hybrid frameworks combining the flexibility of software and the performance of hardware accelerators designed to offload network probes from selected computationally expensive tasks. The design of hybrid frameworks requires expertise in software, firmware and hardware development, as well substantial investments that have a negative impact on end-user prices. In fact, since the target of these devices is a niche market, their price is in order of magnitudes higher than commodity off-the-shelf network interfaces.

Packet capture accelerators are the most cost effective solution for improving software based traffic monitoring applications. As packet capture is the cornerstone of many passive monitoring application, capture accelerators have been able to provide substantial speedups to traffic monitoring applications by allowing incoming traffic to be copied directly into the address space of the analysis process without any CPU assistance.

In our past research, we focused on pure-software traffic analysis frameworks. In particular, we proposed filtering solutions that are capable to overcome the limitations of the popular Berkley Packet Filter (BPF) [8], a rule-based traffic filtering mechanisms provided by the majority of the operating systems. In [9] we describe a traffic filtering mechanism that, contrary to BPF, can be reconfigured in real-time and scale in terms of number of traffic filtering rules. In [10] we present a traffic filtering and analysis framework named RTC-Mon that substantially simplifies the development of modular and efficient traffic monitoring applications. The core of the framework is a rule-based infrastructure that allows traffic analysis components to be enabled over the traffic matching rules. By introducing services for IP de-fragmentation, packet parsing and maintenance of flow state statistics, the development efforts for implementing monitoring applications are substantially reduced. The framework is useful for implementing traffic analysis applications, such as VoIP and IPTV monitoring software, where traffic filters must be added/removed in real-time.

In our previous works, we decided not to leverage any specific monitoring device in order to reduce costs and simplify the deployment. In this work instead, we evaluate the opportunity of accelerating our framework by exploiting mainstream NICs. Unlike special purpose monitoring hardware, off-the-shelf network interfaces target the mainstream market and therefore come at low end-customer prices. Even if these NICs are not designed for accelerating monitoring software but rather tasks as virtualization, some of their features can be successfully exploited for increasing the performance of traffic analysis applications.

Modern off-the-shelf adapters provide several independent RX/TX queues and hardware-based mechanisms such as Receive-Side-Scaling (RSS) that balance network flows among RX queues mapped on processor cores. By splitting the traffic among queues, the workload, both in terms of packet processing and interrupts can be balanced across cores for better exploiting the intrinsic parallelism of modern computing architectures. As of today, the majority of server class adapters in the market are multi-queue enabled and support RSS for splitting the traffic across queues. The main limitation of RSS is that the balancing policy is static hence it cannot be adapted to changing traffic conditions. This represents a serious limitation as workload unbalances correspond to scalability penalties. Even if it is possible to augment RSS with software based traffic balancing policies, this approach is, in practice, unfeasible for high-speed networks as the performance penalty is severe. Therefore, NIC manufacturers are introducing the second generation traffic balancing hardware mechanisms that are dynamically configurable in order to adapt traffic balancing policies to every traffic condition. Although these mechanisms have been introduced for enhancing general purpose networking, we believe that packet filtering will also benefit from these breakthrough balancing technologies, and therefore, the performance gap between special purpose monitoring devices and off-the-shelf network adapters would be reduced.

In this work we present an advanced and yet easy to use open source software framework that leverages the customizable hardware assisted traffic balancing and filtering features introduced in modern NICs. As these filtering features will likely be available in future NIC cards manufactured by various vendors just as happened with RSS, we believe that this work is not limited only to the specific NIC we considered in this paper, but it paves the way to supporting a new family of cheap 10 Gbit (and 40 Gbit in the future) network adapters.

3 Related Work

The industry followed three paths for accelerating software applications by means of specialized hardware while preserving the software flexibility:

- Accelerate the capture process via packet capture accelerators [3, 4] that allow incoming packets to be copied directly to the address space of monitoring applications without any CPU intervention.
- Split the monitoring workload among different network probes using smart traffic balancers [5] so that each probe receives and analyzes a portion of the traffic.
- Run traffic analysis software on programmable network cards based on network processors [6] or massive parallel architectures [7]. Programmable network cards are massive parallel architectures on a NIC. Monitoring applications are implemented in C and executed on these device [7] that run a modified version of

Linux which simplifies the porting of existing applications on top of this special purpose architecture. However, even if they have been able to substantially simplify the development compared to network processors based cards, the porting is still not trivial.

Most general purpose operating systems support rule based filtering mechanisms such as the BPF where filtering expressions are compiled into an intermediate language and interpreted by a virtual machine running at the kernel layer. PF_RING [11] is an advanced network monitoring framework enhancing Linux with more flexible filtering mechanisms implemented in software by means of kernel modules. NetVM [15, 16] is a virtual machine designed to simplify the development and maintenance of complex and yet efficient packet processing applications running on top of heterogeneous network devices. FFPF [17] is an extensible and high-performance packet capture and filtering architecture based on Linux. Contrary to our work, FFPF does not leverage modern multi-core or multi-queue interfaces. The SCAMPI project [18] provides a feature rich monitoring API but it has been designed to run on top of specialized monitoring devices and therefore it yields poor performance when running over commodity hardware. [22] describes a framework for high-speed networks monitoring that provides features such as IP defragmentation and flow reassembly, that relies on a pure-software implementation of a packet scheduling algorithm proposed in [23]. Our work instead, exposes to the software layers an API to design hardware assisted packet schedulers.

Capture accelerators based on FPGA, implement filtering mechanisms at the network layer by means of rule sets (usually limited to 32 or 64) similar to BPF. Filtering runs at wire-speed. As the rule set is not meant to be changed at runtime, its scope of application is drastically limited. Often traffic filtering is used to mark packets and balance them across DMA engines. Traffic balancing policies are similar to RSS and are usually implemented at the FPGA layer and allow the traffic to be split among cores within a multi-core processor. As for traffic filtering, dynamically updating the traffic balancing policies at run-time is in practice unfeasible as a card reconfiguration may require seconds if not minutes.

4 Framework Design

In our past research, we developed an extensible traffic analysis framework implemented under the Linux Kernel called PF_RING [11] which accelerates packet capture and implements packet parsing and filtering by means of dynamically loadable kernel plugins. A user space library called libpfring provides an easy to use API that allows user space applications to interact with the framework.

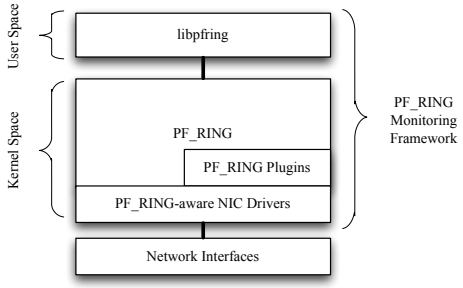

Fig. 1. PF_RING Monitoring Framework.

PF_RING runs on top of commodity network interface cards and can use both standard NIC drivers or PF_RING optimized drivers. These drivers, available for popular 1 and 10 Gbit adapters produced by vendors such as Intel and Broadcom, push incoming packets directly to PF_RING without passing through the standard kernel mechanisms hence accelerating capture speed.

PF_RING provides a flexible rule-based mechanism that allows users to assign packets to kernel plugins which are then responsible to dissect, order in flows, and compute flow metrics (e.g. voice quality) directly at the kernel layer without copying packets to user space. For example, it is possible to configure PF_RING to dispatch TCP packets on port 80 to the HTTP plugin, and UDP packets on port 5060 to the SIP plugin. The same rule-based mechanism can be used for filtering out from PF_RING analysis unwanted packets (e.g. discard packets coming from a specific host or port) similar to what the firewalling layer does at an operating system level.

With the advent of multi-core systems and multi-queue adapters, PF_RING has been extended with support of virtual RX queues [12], that enable specific plugins/user space applications to receive traffic from specific RX queues. The PF_RING kernel infrastructure is responsible to exploit facilities such as RSS for balancing and assigning packets to cores while queue information is preserved in received packets.

In summary, PF_RING has become an advanced framework that thanks to its rule-based mechanism, has been capable to simplify the engineering of modular applications and not just accelerate packet capture. However, the rule-based mechanism has been completely implemented in software, and therefore, it is inefficient at very high speed such as 10 Gbit.

Last year Intel introduced X520, a 10 Gbit card based on the new 82599 ethernet controller [13]. What makes this adapter interesting for PF_RING, is the ability to support in hardware dynamically configurable flow affinity filters for classifying, load balancing and dispatching traffic flows to processor cores. The filtering mechanisms introduced by 82599 can be seen as a fine-grained RSS that allows selected flows to be classified and dispatched towards specific cores based on configurable packet filters and not on RSS hashing.

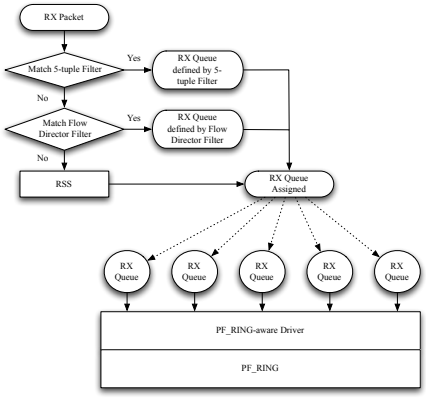

Fig. 2. Integrating 82599 with PF_RING.

The availability of this affinity facility in commodity adapters has been the natural solution to address performance issues of PF_RING at 10 Gbit. Exploiting the flow affinity filters is indeed attractive for:

- leveraging hardware facilities for dispatching packets across PF_RING plugins enabled on selected RX queues;
- dropping unwanted packets in hardware inside the NIC before they hit the driver.

In a nutshell, flow affinity filters introduce new opportunities, not yet exploited by operating systems and monitoring applications, for the implementation of hardware assisted packet schedulers capable to accelerate traffic analysis applications by fully exploiting the parallelism offered by multi-core architectures.

As we believe that 82599 is just the "first of a kind" and similar flow affinity filters mechanisms will soon be introduced by other vendors, PF_RING has been extended not only to exploit these features as implemented by 82599 but also to support future NICs providing similar capabilities. For this reason we introduced a new hardware-neutral software layer that is responsible for setting up specific flow affinity filtering rules in hardware. This layer has not been designed for natively supporting the 82599 controller in PF_RING, but rather as a foundation layer for offloading selected filtering tasks to those NICs that feature flow affinity filters. This means that:

- not all facilities offered by 82599 have been supported yet (e.g. IEEE 1588 time synchronization), but only those (i.e. flow affinity filters) that can be currently exploited by PF_RING for accelerating its operations (i.e. we have not added support of 82599 in PF_RING, but rather exploited those 82599 features that can accelerate PF_RING);
- adding support in PF_RING for flow affinity filters-like features in future NICs, will not require PF_RING redesign but it will just require the implementation of new extensions into PF_RING-enabled NIC drivers;

- existing applications such as RTC-Mon will not need to be recoded (but just slightly modified) in order to exploit flow affinity filters, as PF_RING transparently sets in hardware the appropriate flow affinity filters.

PF_RING supports two families of filters: precise filters where the whole `<vlan, protocol, ip/port src, ip/port dst>` tuple needs to be specified, and wild card filters where some filter parameters can be unspecified (e.g. `tcp and port 80`). When a packet is received, PF_RING uses the "best match first" policy, so it will first try to match the packet against configured precise filters, and in case of no match against wild card filters. Packets matching a filter will be passed to the specified plugin or action, if configured. Hardware flow affinity filters support has been added into PF_RING as follows:

- PF_RING-aware drivers notify (when the driver is loaded inside the kernel) the PF_RING engine whenever a given NIC supports flow affinity filters.
- PF_RING has been extended with a new function named `handle_hw_filtering_rule()` that allow precise and wild carded filters to be added/removed inside NICs.
- For each NIC supporting flow affinity filters, PF_RING adds a virtual file whose path is `/proc/net/pf_ring/ethX/rules` that network administrators, and not just monitoring applications, can use for adding/removing filters by means of a simple echo of a string on it. For instance `echo "+(1,-1,tcp, 192.168.0.10,25,0.0.0.0,0)" > /proc/net/pf_ring/eth3/rules`, instructs PF_RING to add in the eth3 device a new filtering affinity rule with id 1 and that sends all TCP packets from 192.168.0.10:25 to the core id -1. Since the identifier -1 does not correspond to a physical processor core, this rule allows packets matching the filter to be dropped at the NIC layer. Using another existing queue id would simply advise the filtering mechanism to direct the packets to the appropriate queue and hence through the SMP affinity mechanism in the Linux kernel into the desired core.

In order not to modify the existing driver structure by introducing new hooks for adding and removing filters, we decided to jeopardize some existing driver hooks. The advantage is that all current drivers do not need to be changed, and this gives us a way to migrate towards packet filtering integration when supported in Linux[1]. The data structure used to pass filter specifications to drivers is generic and does not rely on 82599 specific data types. In this way, the efforts for supporting future network adapters providing similar features will be substantially reduced. 82599 provides several types of filters including layer 2 and FCoE (Fibre Channel over Ethernet), but

[1] In kernel 2.6.34 the ethtool, not the kernel itself, introduced limited support for EFD thanks to patches we submitted to Linux kernel maintainers.

as PF_RING supports only precise and wild card filters, we focus only on 5-tuple and flow director filters that are very close to PF_RING filters:

- 5-tuple filters (up to 128 filters can be defined in 82599) allow packets belonging to flows identified by the 5-tuple <protocol, ip source, port source, ip destination, port destination> to be forwarded to a specific RX queue. 5-tuple filters are defined as `<id, protocol, ip/port src, ip/port dst, target RX queue id>`. Some of the fields specified in a 5-tuple filter can be "masked" (i.e. wild carded) in order to avoid comparing them against incoming packets.
- Flow Director (FD) filters can be specified as precise (i.e. the filter members are matched precisely against incoming packets) or hash (i.e. the packet hash is compared against the filter hash, conceptually similar to bloom filters [14]) filters. 82599 supports up to 32k precise filters. The number of distinct hash filters is not limited by design. However, the adoption of excessive hash filtering rules may lead to false positives. FD filters are expressed as `<slot id, VLAN, protocol, ip netmask/port src, ip netmask/port dst, target RX queue id>`. Currently all configured filters must have the same mask defined in 82599.

The 82599 adapter is quite different from many FPGA-based NICs as it does not use a TCAM (Ternary Content Addressable Memory) for handling filters. This means that a filter is configured by setting up specific NIC registers and, therefore, that the last configured filter overwrites the previous register value. For this reason, it is not possible to read from the NIC all configured filters, and therefore the driver has to maintain the list of configured filters. The advantage of this approach is that, contrary to many FPGA-based NICs where setting a filter requires card reconfiguration, in 82599 setting a filter is extremely fast and from the application point of view it takes as long as the `setsockopt()` system call necessary to pass the filter specification to the kernel, making this NIC usable in environments where filter configuration has to be dynamically changed.

5 Use Cases and Validation

Validation has been performed using an IXIA XM12 10 Gbit traffic generator and a NUMA computer using a single 6-core Xeon® X5650 (Westmere) CPU at 2.67GHz. In all tests we have injected IPv4 UDP traffic with random payload at wire speed, and compared the number of packets sent by the traffic generator with those reported by *pfcount*, a simple packet-counting application running on top of PF_RING. pfcount spawns and binds a thread per core (i.e. thread X is bound to core X). The injected traffic contained 6 flows, each balanced to an individual core using hardware filtering

rules. Packets have been captured using the standard NAPI-based 82599 driver enhanced with PF_RING and hardware filtering support.

Table 1. Hardware vs. Software Filtering Comparison

Frame Size (Bytes)	Test 1		Test 2	
	Software Filter (Capture Rate)	Hardware Filter (Capture Rate)	Software Filter (CPU Load)	Hardware Filter (CPU Load)
64	5.7%	6.3%	95.6%	None
128	10.0%	11.6%	95.4%	None
256	19.5%	23.2%	98.7%	None
512	37.4%	42.3%	3.5%	None
1024	99.8%	100%	3.3%	None
1518	99.6%	100%	< 0.1%	None

In the first test we compared hardware (i.e. 82599) vs. software (i.e. PF_RING) packet filtering using a single filtering rule that match for every incoming packet (i.e. the entire traffic is forwarded to the user space). In the second test we have injected traffic that does not match any configured filter, and verified that there is no load on the CPU whenever hardware filters are used. On the contrary, what we observed with software filters, is that for packets up to 256 bytes the CPU utilization was around 95%, and about 3% for larger packets. This leads us to the conclusion that in the hybrid model of software and hardware filtering we propose, it is recommended to use software filters only for medium to large packets.
In order to further improve packet capture, the authors have developed TNAPI [25], a multithreaded RX queue polling mechanism that significantly improves packet capture performance with respect to the standard Linux NAPI.

5.1 Realtime Multimedia Traffic Monitoring

As described earlier in this paper, RTC-Mon has been designed to efficiently handle VoIP calls and video-on-demand traffic analysis at 1 Gbit. In order to scale the solution to 10 Gbit, we have slightly modified the original RTC-Mon code as follows:

- A few 5-tuple filters have been configured:
  - All the SIP signaling packets go to core 0.
  - Non UDP (i.e. ICMP/TCP) packets are dropped.
  - UDP traffic on popular ports (e.g. port 53 used by DNS) is also dropped.
- Whenever a new VoIP call has been setup, such call is tracked by adding two FD filters (one per call direction) that send the voice traffic for the tracked call (i.e. RTP traffic) to the same RX queue where the RTP plugin is active. In order to evenly balance the traffic across queues, the queue ids used for voice traffic are selected in round robin so that all queues have almost the same amount of traffic.

This setup has allowed RTC-Mon to operate efficiently in 10 Gbit links where VoIP is only a portion of the overall traffic, thanks to 82599 filters used to discard packets not belonging to calls being tracked. Unfortunately, not all unwanted packets have been discarded and a small portion of them is still received by PF_RING. This is because 5-tuple filters are evaluated before FD filters, hence it is not possible to set 5-tuple rule that discards all the remaining traffic because this would also discard traffic that matched by FD filters. It is worth noting that the ability to setup thousands of flow affinity filters with almost no latency is a key factor for using effectively 82599 in cases where filter setup latency is crucial as with RTC-Mon.

5.2 Network Troubleshooting

Troubleshooting a heavily loaded 10 Gbit link using popular tools such as tcpdump and wireshark [24] is almost impossible due to severe packet capture loss. Furthermore, most commercial tools are not distributed with source code, hence it is not possible to recompile them in order to take advantage of PF_RING flow affinity filters. In this case, we used PF_RING's `/proc` interface for setting a few traffic filtering rules that discard in hardware unwanted traffic, hence pass to the Linux kernel only those packets that must reach network monitoring applications. This solution has the advantage that existing applications do not need to be modified, and PF_RING is used just for allowing the network administrator to easily configure (e.g. using a shell script) flow affinity filters without having to code a C/C++ application sitting on top of libpfring.

5.3 Traffic Classification and Balancing

In case monitoring applications do not run on the same box where an 82599 based NIC is installed (e.g. because they run on a non-Linux OS such as Windows), it is possible to create a traffic filtering box using the *pfreflect* application part of PF_RING, that filters incoming packets and copies them onto one or more NICs based on the PF_RING filters configuration. As PF_RING filters (hence flow affinity filters) are evaluated before reflection (i.e. packet bridging in PF_RING parlance), this application can be used for creating an inexpensive traffic filtering box that can be used for reducing the amount of traffic to analyze. If the filtered traffic is less than one Gbit it can be forwarded onto a 1 Gbit card so that legacy measurements box do not need to be updated to 10 Gbit. Furthermore as PF_RING supports traffic balancing, it is possible to forward filtered traffic onto several output interfaces by balancing each RX queue of 82599 onto a different output interface. This solution allows high-speed links to be monitored and troubleshooted without having to purchase costly 10 Gbit measurement boxes.

5.4 Lawful Interception of Internet Traffic

Since the approval of the wiretapping in the US in 1984, lawful interception (LI) has become very popular. In LI a lawful authority requires to intercept and store specific traffic for the purpose of analysis or evidence. In IP networks, this means that traffic originated/directed to specific IPs or flowing on specific ports need to analyzed. Doing this on a 10 Gbit link using software-based traffic filters can be inefficient as packet loss might prevent captured traffic from being analyzed properly. In order to implement a simple packet capture system driven by signaling protocols such as Radius or DHCP, it is possible to setup (e.g. via the PF_RING /proc filesystem interface) a few filtering rules that discard all traffic except signaling (similar to the setup used in 5.2) and traffic belonging to target IPs that need to be intercepted.

5.5 Firewalling at 10 Gbit

The Linux netfilter/iptables firewall is quite efficient but it cannot operate with no loss on heavily loaded 10 Gbit links. The use of 5-tuple filters can definitively help dropping unwanted traffic or tracking NAT sessions using FD filters. Unfortunately the Linux firewall is more flexible than 5-tuple filters, hence it is not possible to do a one-to-one mapping between iptables rules and 5-tuple filters. This means that 82599 can be used to discard a large portion of incoming traffic but not all, leaving to netfilter the duty of completing packet filtering. Nevertheless this hybrid, hardware plus software, filtering architecture allows to significantly boost the firewall performance in most situations. Currently we are add filters using the PF_RING /proc filesystem interface as we have not yet added native 82599 support into netfilter.

6 Open Issues and Future Work

The main limitation of the current implementation is the lack of a compiler that transparently compiles BPF filters into PF_RING (hence flow affinity) filters. Due to this limitation, users must configure both BPF filters (e.g. on the command line while starting the monitoring tool) and flow affinity filters (e.g. using the PF_RING `/proc` filesystem). In future code releases we plan to implement such feature so that BPF-aware applications (e.g. Wireshark) can still use BPF for setting filters while the underlying kernel layers add automatically flow affinity filters in order to reduce the amount of packets that will hit the BPF filtering engine. In addition to 5-tuple and FD filters, 82599 also supports SYN filter that diverts to a specific core all incoming TCP packets with the SYN flag set. While its support would be trivial from the 82599 point of view, the PF_RING engine instead needs some extensions in order to add filters that can select packets based on TCP flags.

Finally we would like to use 82599 in the context of OpenFlow switching, for implementing efficient in-kernel switching across network applications without requiring external switching equipment. From the hardware point of view, we envisage that future NICs will further enhance flow affinity filters number and expressiveness (e.g. adding the ability to filter tunneled traffic), add per-filter statistics (e.g. number of packets and bytes that matched each filter) so that developers could implement efficient NetFlow caches in hardware.

7 Conclusions

Monitoring the Internet is challenging as high-speed networks are becoming popular and traffic patterns more complex. In order to satisfy the increasing performance requirements and reduce deployment costs, modern network monitoring frameworks should leverage those features offered by mainstream NICs that are introduced for general-purpose networking and not fully exploited in the context of network monitoring. This paper has presented an evolution of PF_RING, a monitoring framework originally designed for accelerating packet capture, that exploits hardware-based filtering mechanisms offered by the Intel 82599 based NICs and likely future NICs. Thanks to flow affinity filters PF_RING can now fine-grain flow balance packets across cores, classify traffic and discard unwanted communication patterns directly into the NIC before packets hit the driver. The validation process has demonstrated that many network applications can benefit from this work, making it very general and usable also outside of the network monitoring domain. Not to mention that it is finally possible to combine the speed of hardware with the flexibility of software for effectively monitoring 10 Gbit networks using commodity network adapters.

Availability. This work is distributed under the GNU GPL license and is available at no cost form the PF_RING home page (http://www.ntop.org/PF_RING.html).

Acknowledgments. The authors would like to thank Intel and in particular Edward Clinton and Richard P. Kelly for their support during this research work.

References

1. W. John and others, Passive internet measurement: Overview and guidelines based on experiences, Computer Communications, vol. 33, issue 5 (2010).
2. G. Memik and W.H. Mangione-Smith, Specialized Hardware for Deep Network Packet Filtering, Proc. of FPL 2002, Montpellier, France, (2002).
3. S. Donnelly, DAG Packet Capture Performance, White Paper, (2006).

4. Napatech Inc, The Napatech Protocol and Traffic Analysis Network Adapter, White Paper, (2006).
5. cPacket Networks, cVu 320G: Aggregation, Complete Packet Inspection Filtering, Automatic Flow Balancing, (2010).
6. P. Crowley and others, Characterizing processor architectures for programmable network interfaces, Proc. of the 14th international conference on Supercomputing, Santa Fe, New Mexico, (2000).
7. A. Agarwal, The Tile processor: A 64-core multi-core for embedded processing, Proc. of HPEC Workshop, (2007)
8. S. McCanne and V. Jacobson, The BSD packet filter: A new architecture for user-level packet capture, Proc. of Winter '93 USENIX Conference, (1993).
9. L. Deri, High-Speed Dynamic Packet Filtering, Journal of Network and System Management, (2007).
10. F. Fusco and others, Enabling High-Speed and Extensible Real-Time Communications Monitoring, In Proc of IM 2009, (2009).
11. L. Deri, Improving Passive Packet Capture: Beyond Device Polling, Proc. of SANE 2004, (2004).
12. L. Deri, Towards 10 Gbit NetFlow Monitoring Using Commodity Hardware, Proc. Joint EMANICS/IRTF-NMRG Workshop, Munich, (2008).
13. Intel Corporation, 82599 10 GbE Controller Datasheet, Rev. 2.3, (2010).
14. B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, July 1970.
15. F. Risso and others, Extending the NetPDL language to support traffic classification, Proc. of IEEE Globecom, (2007).
16. L. Degioanni and others, Network virtual machine (NetVM): a new architecture for efficient and portable packet processing applications, Proc. of 8th International Conference on Telecommunications, (2005).
17. H. Bos and others, FFPF: Fairly fast packet filters, Proc. of OSDI '04, (2004).
18. J. Coppens and others, SCAMPI: A Scalable and Programmable Architecture for Monitoring Gigabit Networks, Proc. of E2EMON Workshop, (2003).
19. L. Deri, nCap: Wire-speed Packet Capture and Transmission, Proc. of E2EMON, (2005).
20. L. Degioanni and G. Varenni, Introducing Scalability in Network Measurement: Toward 10 Gbps with Commodity Hardware, Proceedings of IMC '04, (2004).
21. M. Smith and others, Enabling High-Performance Internet-Wide Measurements on Windows, Proc. of PAM 2010, pp. 121-130, Zurich, Switzerland, (2010).
22. M. Dashtbozorgi and others, A scalable multi-core aware software architecture for high-performance network monitoring, Proc. of the 2nd international conference on Security of information and networks, pp. 117-122, Famagusta, Cyprus, (2009).
23. M. Dashtbozorgi and others, A high-performance software solution for packet capture and transmission, Proc. of 2nd IEEE International Conference on Computer Science and Information Technology, pp. 407-411, Beijing, China, (2009).
24. F. Fuentes and D. C. Kar, Ethereal vs. Tcpdump: a comparative study on packet sniffing tools for educational purpose, Journal of Computing Sciences in Colleges, Vol. 20, Issue 4, pp. 169 - 176 , (2005).
25. L. Deri and F. Fusco, Exploiting Commodity Multi-core Systems for Network Traffic Analysis, Technical Report, http://luca.ntop.org/MulticorePacketCapture.pdf, (2009).