Improving Passive Packet Capture: Beyond Device Polling

Luca Deri

Packet Capture: State of the Art

- Many available passive network monitoring tools are based on libpcap (http://www.tcpdump.org) or winpcap (http://winpcap.polito.it)
- Despite libpcap offers the very same programming interface across different OSs, the library performance are very different depending on the platform being used.
- Some library components (e.g. BPF packet filtering) have been implemented into the kernel for better performance.

Packet Capture: Open Issues

- Monitoring low speed (100 Mbit) networks is already possible using commodity hardware and tools based on libpcap.
- Sometimes even at 100 Mbit there is some (severe) packet loss: we have to shift from thinking in term of speed to number of packets/second that can be captured analyzed.

Problem Statement

- Ability to monitor high speed (1 Gbit and above) networks with common PCs (64 bit/66 Mhz PCI bus) without the need to purchase custom capture cards or measurement boxes.
- Have spare CPU cycles for traffic analysis.

Libpcap Performance (Polling)

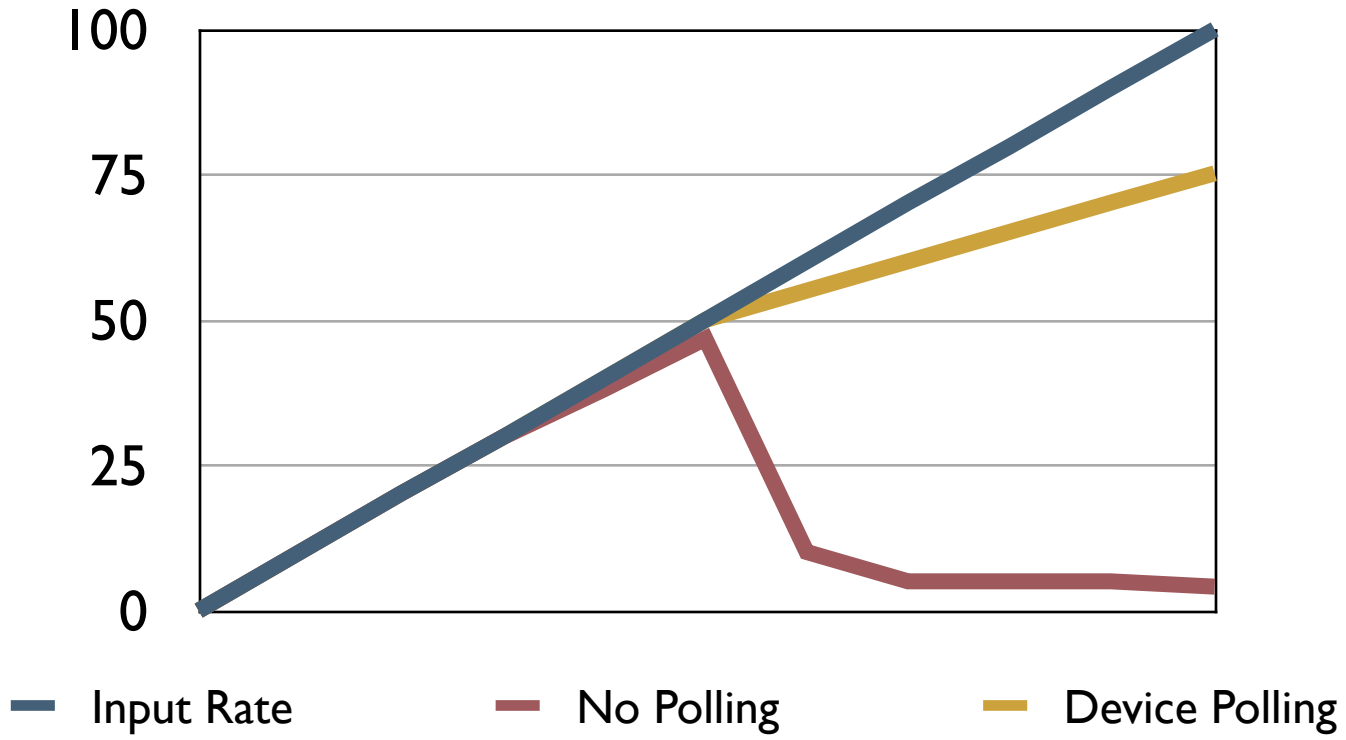
Packet Size (Bytes)	Speed (Mbit)	Speed (Pkt/sec)	Linux 2.6.1 with NAPI and standard libpcap	FreeBSD 4.8 with Polling
64	90	175'000	2.5%	97.3%
512	710	131'000	1.1%	47.3%
1500	836	70'000	34.3%	56.1%

Percentage of captured packets

Testbed:

- Sender: Dual 1.8 GHz Athlon, 3Com 3c59x (100Mbit) Ethernet card
- Receiver: Pentium III 550 MHz, Intel 100Mbit Ethernet card
- Traffic Generator: stream.c (DoS)

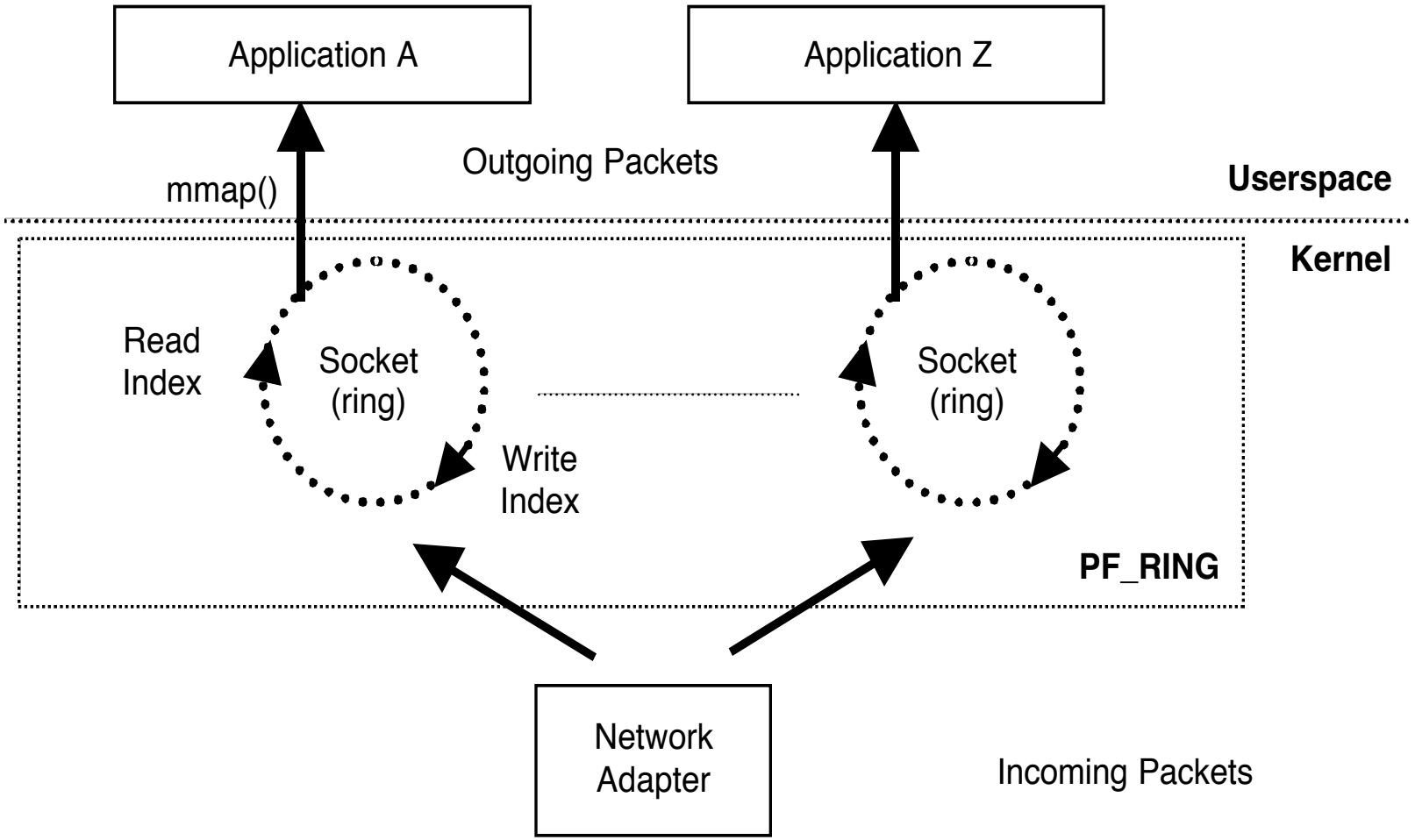
Polling vs. non Polling



Sidenote. Polling is usually disabled by default in OSs as:

- Slow polling can have negative effects on packet timestamp precision.
- Fast polling can take over all/most of the available CPU cycles.

Our Solution: Socket Packet Ring (PF_RING)



Driver Packet Ring: Features [1/2]

-
- The new ring socket is right on top of the netif_rx/ netif_prepare_rx routines: no driver changes are needed.
- Features like packet sampling and kernel filtering can be efficiently implemented with the ring (otherwise the packet arrives to the kernel/userland then is filtered/sampled).
- The bucket copy operation is very fast (via memcpy() with no memory handling needed) so even with kernel that does not support device polling, under strong traffic conditions the system is usable.

Driver Packet Ring: Features [2/2]

- Straight path from the driver to userspace with no kernel overhead (beside the PCI bus). Note that packets are transfer over the PCI bus via DMA and that interrupts arrive only when the packet is already available to the driver.
- Packets are not queued into kernel network data structures but directly accessible to user-space applications.
- The mmap() primitive allows userspace applications to access the circular buffer with no overhead due to system calls such as in the case of socket calls.

Driver Packet Ring: Advantages

- Enable people to capture packets at (almost if not full) wire speed without having to purchase a dedicated card or measuring box.
- It exploits both polling and interrupt mitigation (when available).
- Common belief that coding into the kernel (on a vanilla system) is more efficient than in userspace (with PF_RING) is not completely correct. In fact a kernel module is slower than a userland application+PF_RING as packets have to traverse kernel structures until they can talk with the kernel module.

Packet Capture Evaluation [1/2]

Packet Size (Bytes)	Speed (Mbit)	Speed (Pkt/sec)	Linux 2.6.1 with NAPI and standard libpcap	Linux 2.6.1 with NAPI and mmap()	Linux 2.6.1 with NAPI and Ring	FreeBSD 4.8 with Polling
64	90	175'000	2.5%	14.9%	75.7%	97.3%
512	710	131'000	1.1%	11.7%	47%	47.3%
1500	836	70'000	34.3%	93.5%	92.9%	56.1%

Testbed:

- Sender: Dual 1.8 GHz Athlon, Intel GE 32-bit Ethernet card
- Collector: Pentium III 550 MHz, Intel GE 32-bit Ethernet card
- Traffic Generator: stream.c (DoS)

Packet Capture Evaluation [2/2]

Evaluation:

- Major improvement with respect to Linux with NAPI
- It can exploit both polling and Linux 2.4.x/2.6.x

Open Issues

- Packet loss: still too many packets are lost on Linux, both in general and also compared to FreeBSD.
- CPU Usage: on Linux there still an important packet loss although the CPU usage is very low (<< 30% on Linux, ~100% on FreeBSD).

IRQ and CPU Load

- Doing some measurements with rdtsc it seems that when there is too much work (e.g. high traffic), the Linux kernel wastes a lot of time with interrupts and spinlocks (kernel mutexes).
- System calls (e.g. poll() used by userland applications to wait for new incoming packets) are too slow. For instance a dummy poll() call takes more than 1'000 CPU cycles (up to ten times as much with high load).
- Further improving packet capture is not the way to go as the problem is the kernel.
- It is necessary to change the kernel for implementing predictability, namely for adding the ability to determine task completion with some time constraints.
- Solution: add realtime-like support on Linux via rtirq patch.

PF_RING and RTIRQ

Packet Size (Bytes)	Linux 2.4.23 with NAPI, RT_IRQ and Ring (Pkt Capture)	Linux 2.4.23 with NAPI, RT_IRQ and Ring (nProbe)
64	550'789 [~202 Mbit]	376'453 [~144 Mbit]
512	213'548 [~850 Mbit]	213'548 [~850 Mbit]
1500	81'616 [~970 Mbit]	81'616 [~970 Mbit]

Captured Packets and nProbe Flow Generation (packet/sec)

Testbed:
Sender: Dual 1.8 GHz Athlon, Intel GE 32-bit Ethernet card
Collector: Pentium IV 1.7 GHz, Intel GE 32-bit Ethernet card
Traffic Generator: stream.c (DoS)

PF_RING: Performance

Product	NetFlow Performance	Price
Cisco Router with MSFC-2	150'000 pkt/sec (on a Cisco 7500)	46'000 $ (MSFC-2 Only)
Juniper Router with PM-PIC	250'000 pkt/sec	~30'000 $ (PM-PIC Only)
PF_RING (Pentium IV 1.7 Ghz)	376'000 pkt/sec	Free

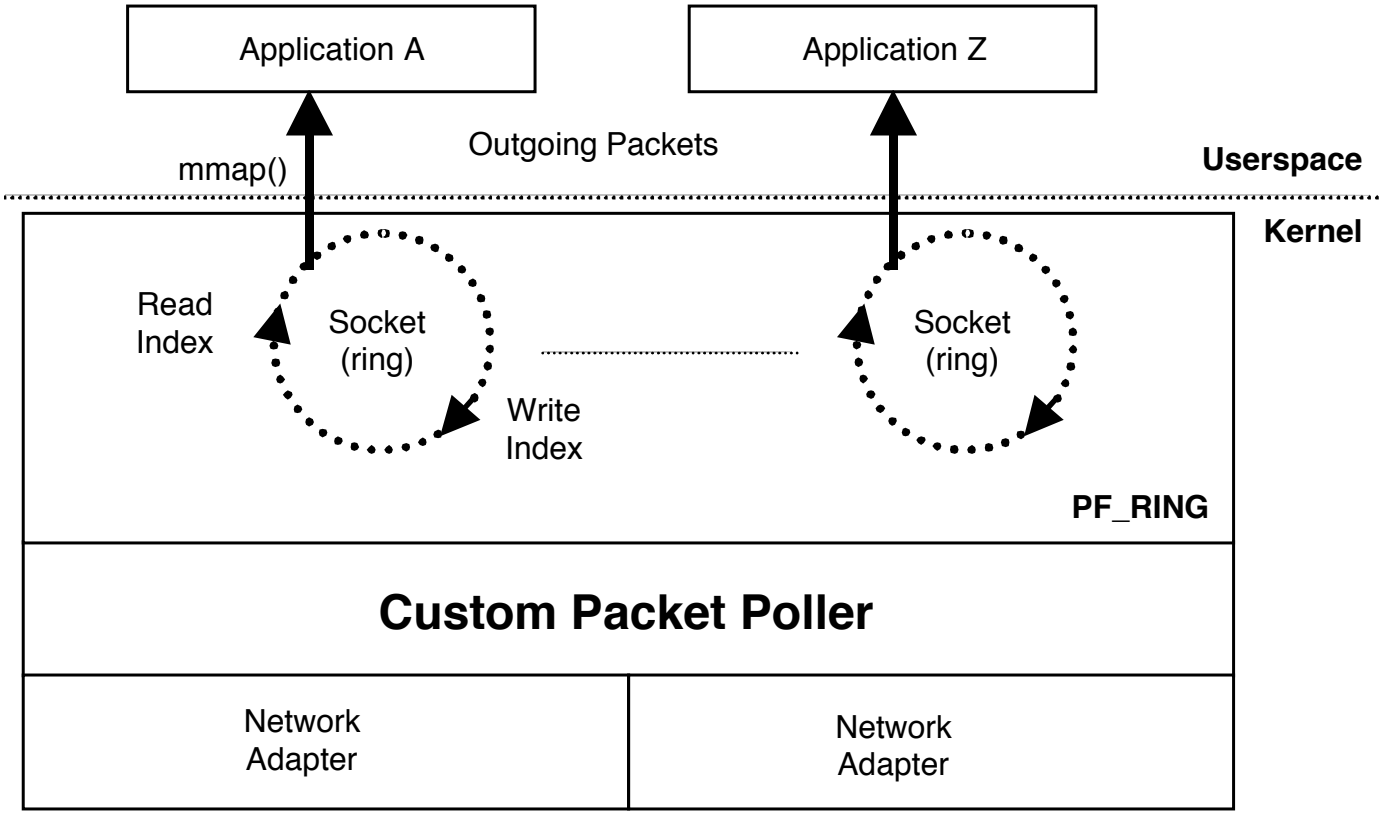
Current Status [1/2]

- Faster flow generation than every commercial NetFlow probe
- Available both on Linux 2.4.x and 2.6.x
- Enhanced libpcap to seamlessly support the ring.
- Successfully tested several applications on top of the ring in order to validate the code.
- ntop and nProbe (homegrown NetFlow v5/v9 probe for IPv4/6), as well as snort/tcpdump are able to run on top of the ring.

Current Status [2/2]

- Code freely available on the Internet.
- PF_RING issues discussed on an ad-hoc mailing list (~40 subscribers).
- Several problems fixed thanks to the "distributed testing" (the GPL way).
- Some companies are using it for real network monitoring of backbones.

PF_RING Spinoffs [1/3]



PF_RING Spinoffs [2/3]

Approach	Packet Poller
Classic PF_RING	netif_rx/netif_receive_rx
O. Drain (Cornell)	On-card poller (National Semiconductor 83820)
L. Degioanni (Polito)	1-CPU Polling [Linux Processor Affinity]

PF_RING Spinoffs [3/3]

Approach	Packet Poller
Classic PF_RING	650 K pkt/sec [No Interrupt Mitigation]
L. Degioanni (Polito)	1.1 Million pkt/sec [1 CPU @ 100%, 3 CPU @ ~0%]

Testbed:

- Collector: Dual 2.4 GHz Xeon (2 Virtual + 2 Physical CPUs), Intel GE PCI-X
- OS: Linux 2.6.x with NAPI
- Packet Size: 60 bytes

Final Questions

- Are we sure we still need custom hardware for high speed traffic analysis when you can capture over a million packets with no loss? Isn't that this problem has been finally solved for the current network generation?
-
- Isn't that we now have a different problem? Namely isn't it time to (finally) develop monitoring applications able to do something really smart with captured packets?

Availability

- Paper and Documentation: http://luca.ntop.org/Ring.pdf
- Code (GNU GPL2 license): http://sourceforge.net/projects/ntop/