Linux Day 2016

High-speed packet processing made easy

Alfredo Cardigliano <cardigliano@ntop.org>

ntop

LINUX DAY

Introduction

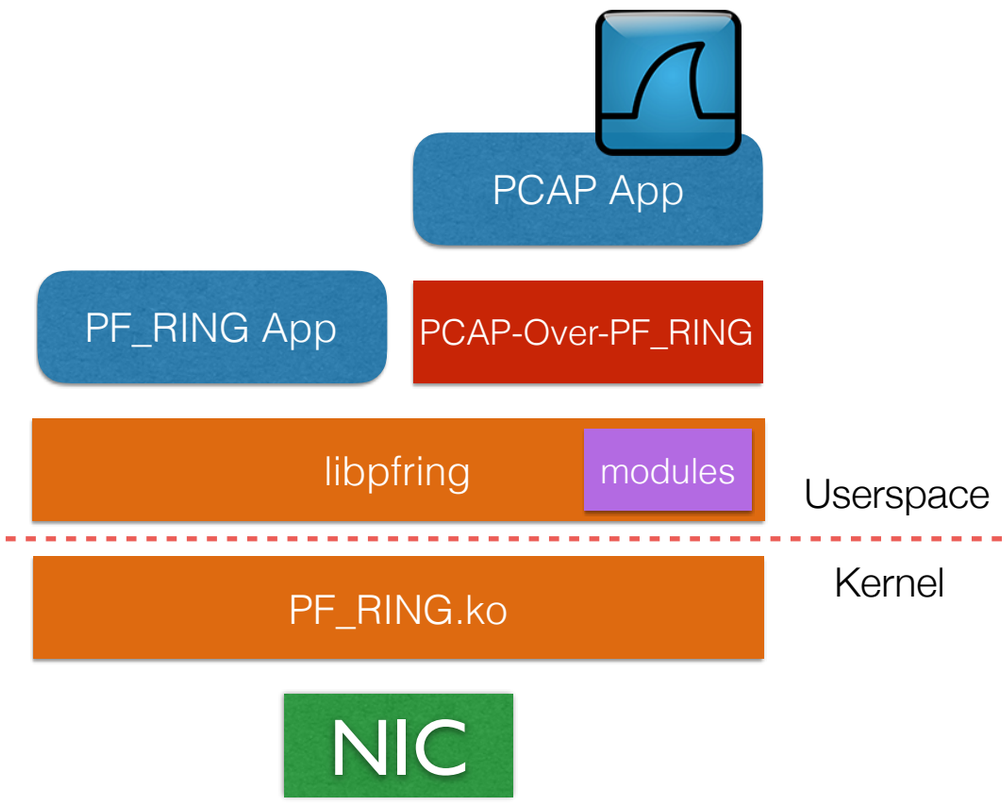
- Network Monitoring tools need raw, high-speed, promiscuous packet capture.
- Commodity network adapters and device drivers are designed for providing host connectivity.

PF_RING

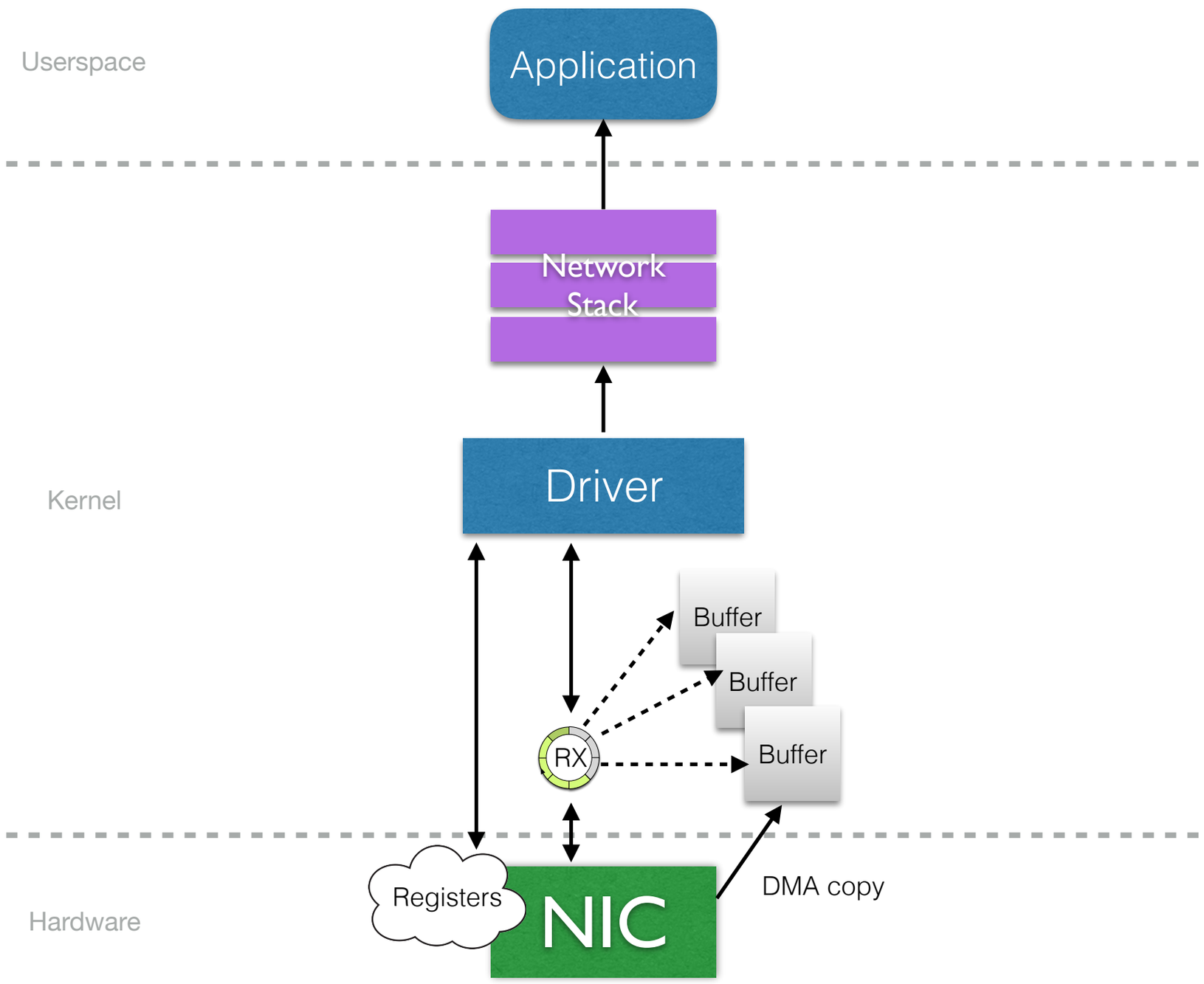
- Open source packet processing framework for Linux.
- Originally (2003) designed to accelerate packet capture on commodity hardware, using patched drivers and in-kernel filtering.
- Today it supports almost all Intel adapters with kernel-bypass zero-copy drivers and almost all FPGAs capture adapters.

PF_RING Architecture

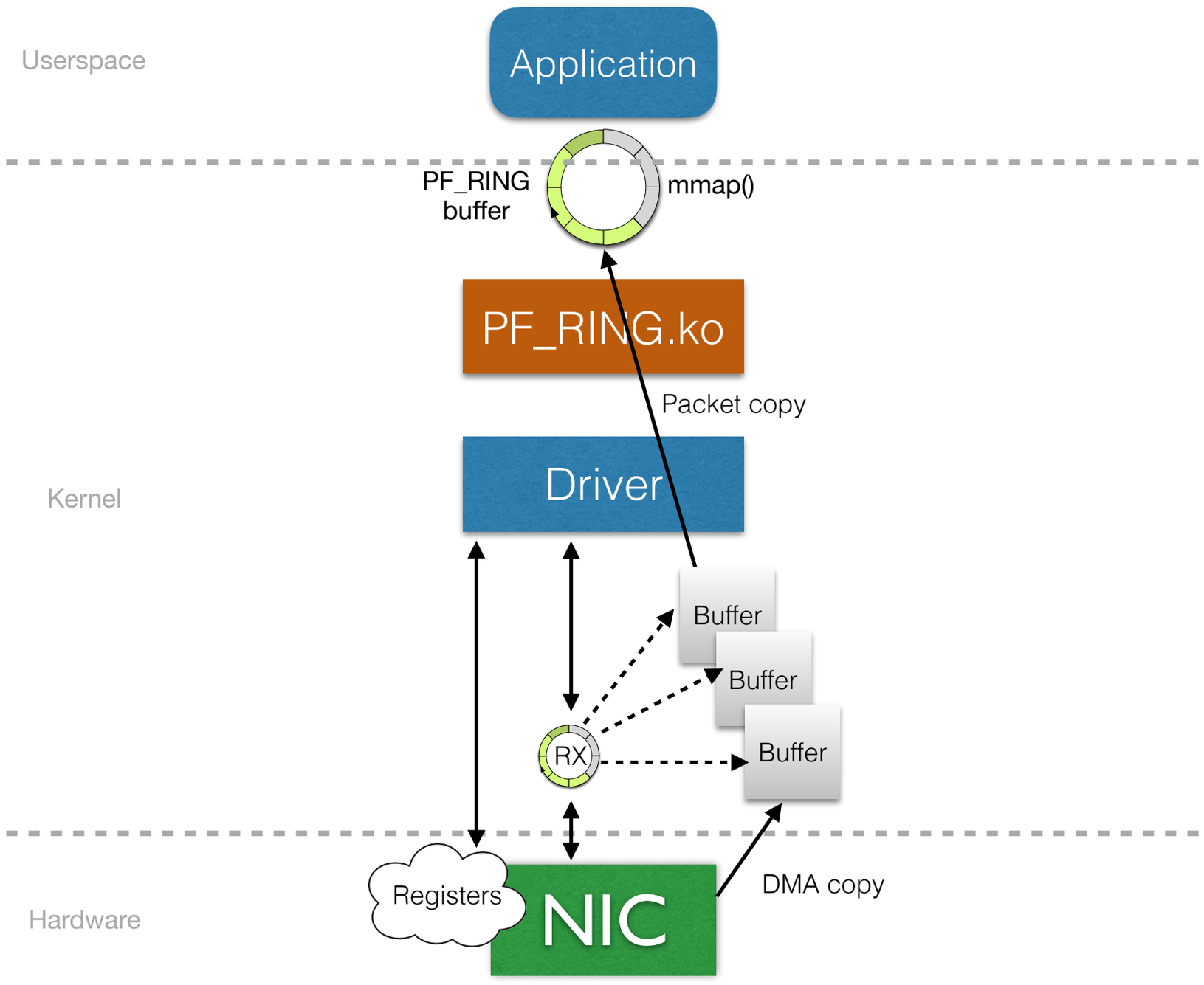
- PF_RING consists of:
  - Kernel module (pf_ring.ko)
  - Userspace library (libpfring)
  - Userspace modules implementing multi-vendor support
  - Patched libpcap for legacy applications



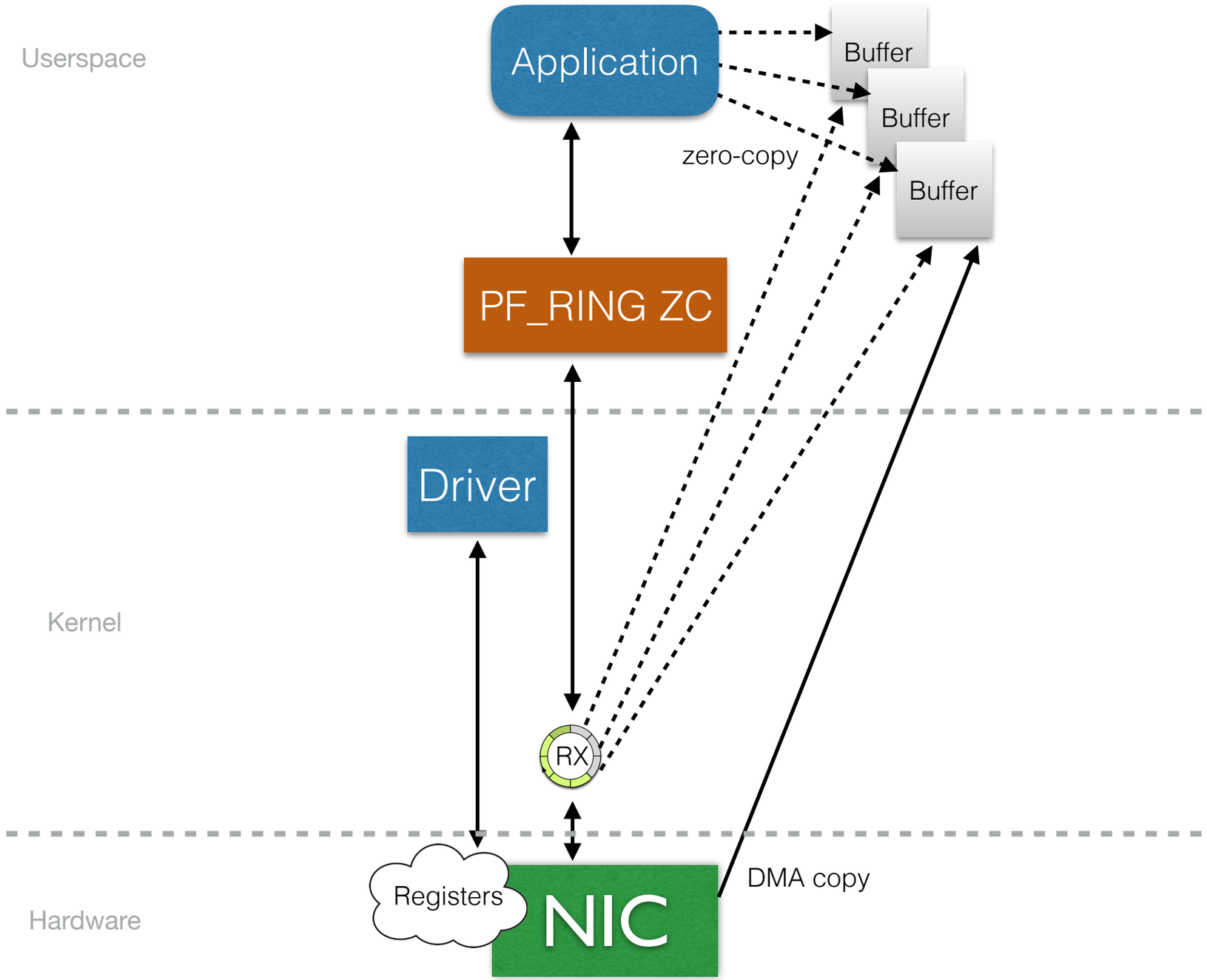
Network Drivers



PF_RING Standard Drivers

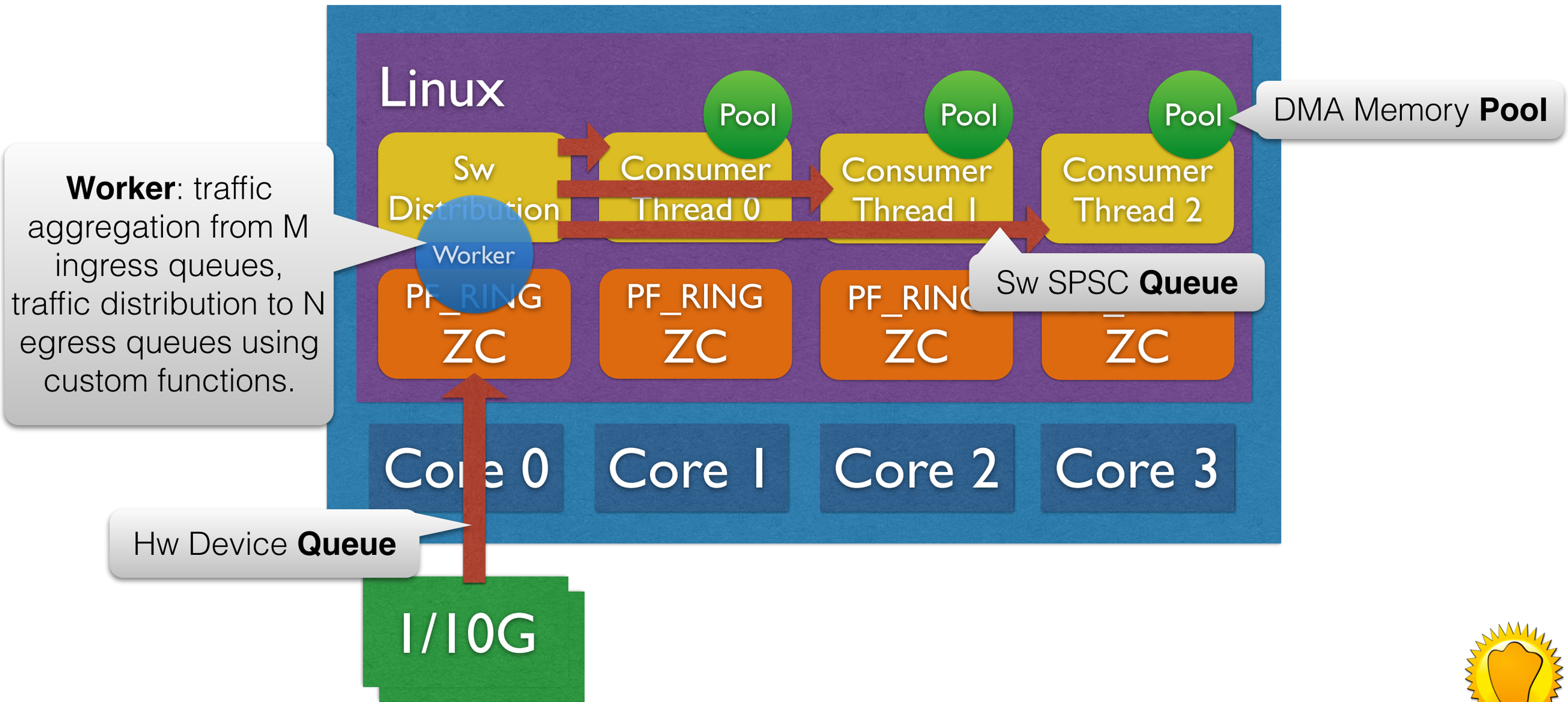


PF_RING Zero-Copy Drivers (Kernel-Bypass)



PF_RING Zero-Copy API

- PF_RING ZC is not just a zero-copy driver, it provides a flexible API for creating full zero-copy processing patterns.



PF_RING ZC API - Example

```
1 zc = pfring_zc_create_cluster(ID, MTU, MAX_BUFFERS, NULL);
2 for (i = 0; i < num_devices; i++)
3  inzq[i] = pfring_zc_open_device(zc, devices[i], rx_only);
4 for (i = 0; i < num_slaves; i++)
5   outzq[i] = pfring_zc_create_queue(zc, QUEUE_LEN);
6 zw = pfring_zc_run_balancer(inzq, outzq, num_devices,
          num_slaves, NULL, NULL, !wait_for_packet, core_id);
```

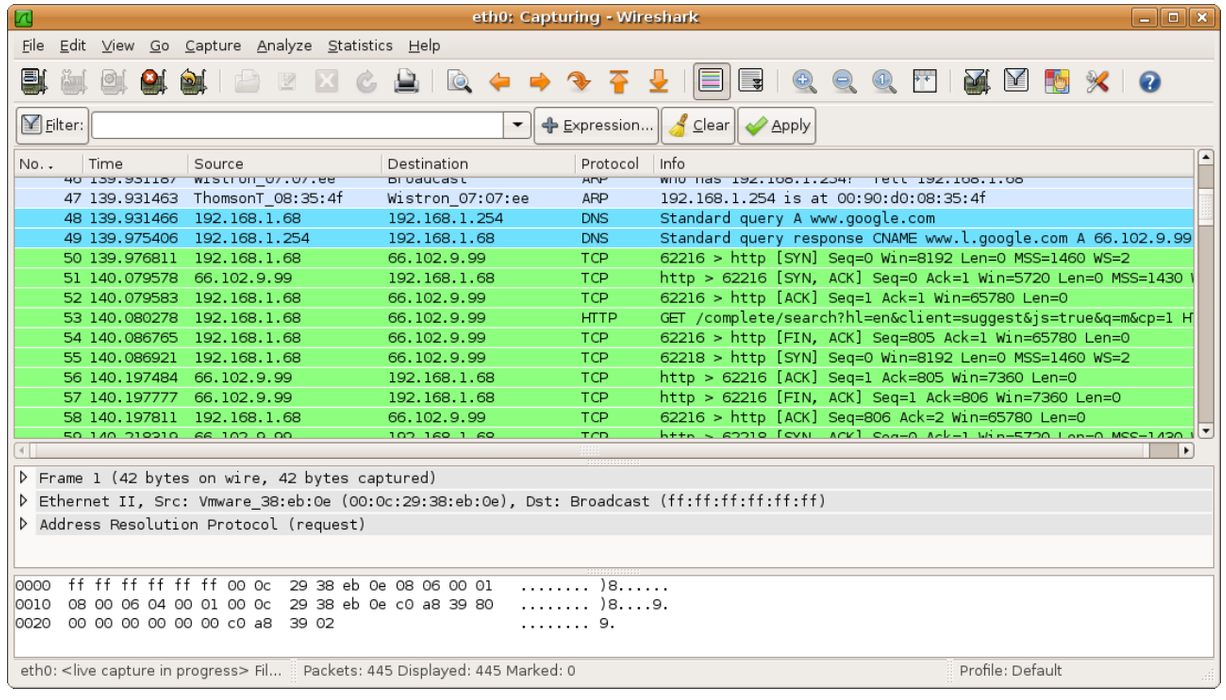
FPGAs

- PF_RING natively supports the following vendors (1/10/40/100 Gbit)

- PF_RING-based applications transparently select the module by means of the interface name:
  - `pfcount -i eth1` [Vanilla Linux adapter]
  - `pfcount -i zc:eth1` [Intel ZC drivers]
  - `pfcount -i nt:1` [Napatech]
  - `pfcount -i myri:1` [Myricom]
  - `pfcount -i exanic:0` [Exablaze]

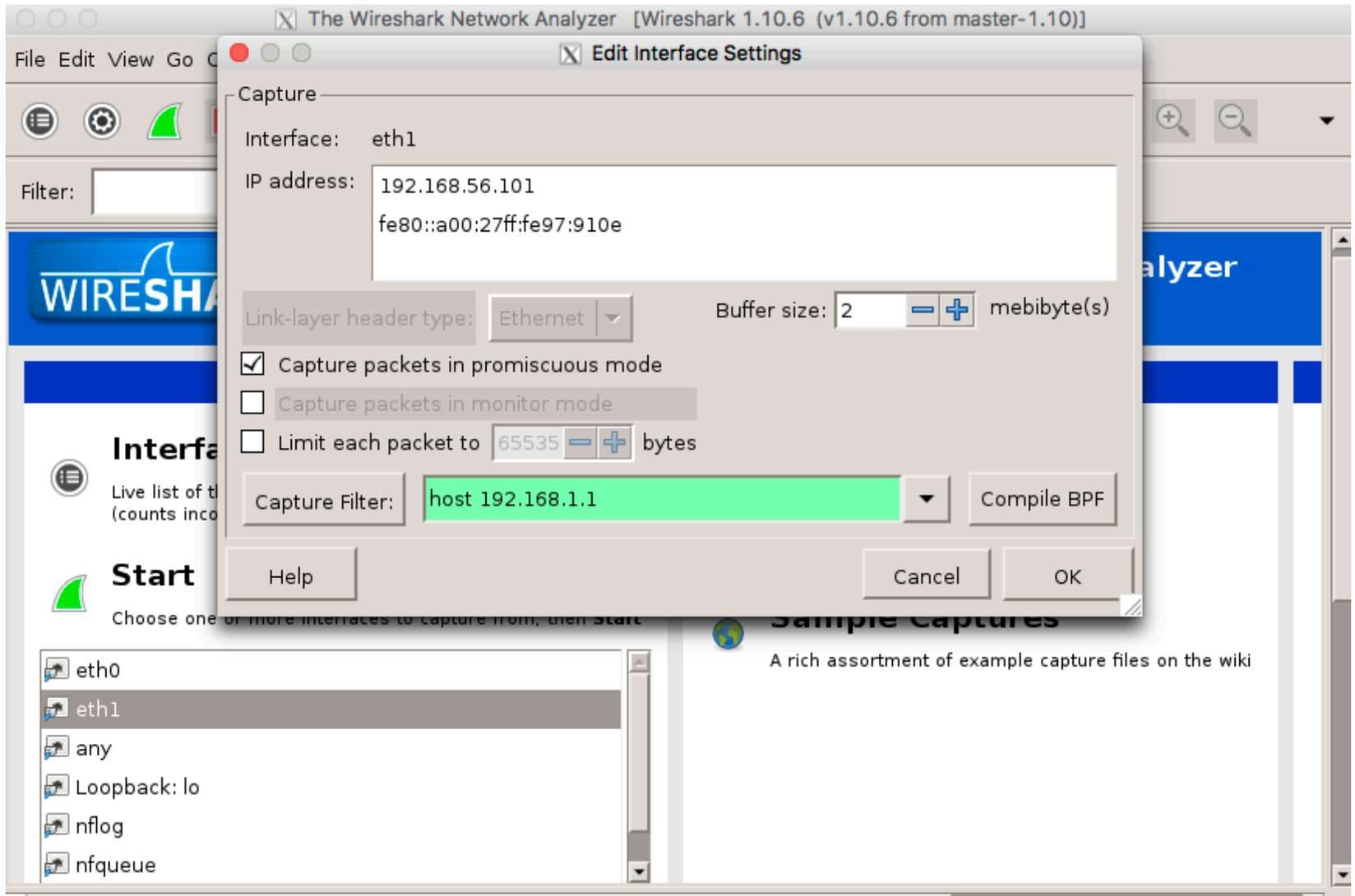
Packet Analysis

- We are accelerating traffic analysis, we are able to process huge amount of traffic and provide aggregated data, what about *packet* analysis?
- We have solutions for capturing up to 100 Gbit line-rate, but what if we want to use Wireshark (or tshark/tcpdump/etc.)?
- Over 1 Gbit using Wireshark on live traffic is challenging due to the number of ingress packets.



Packet Filtering

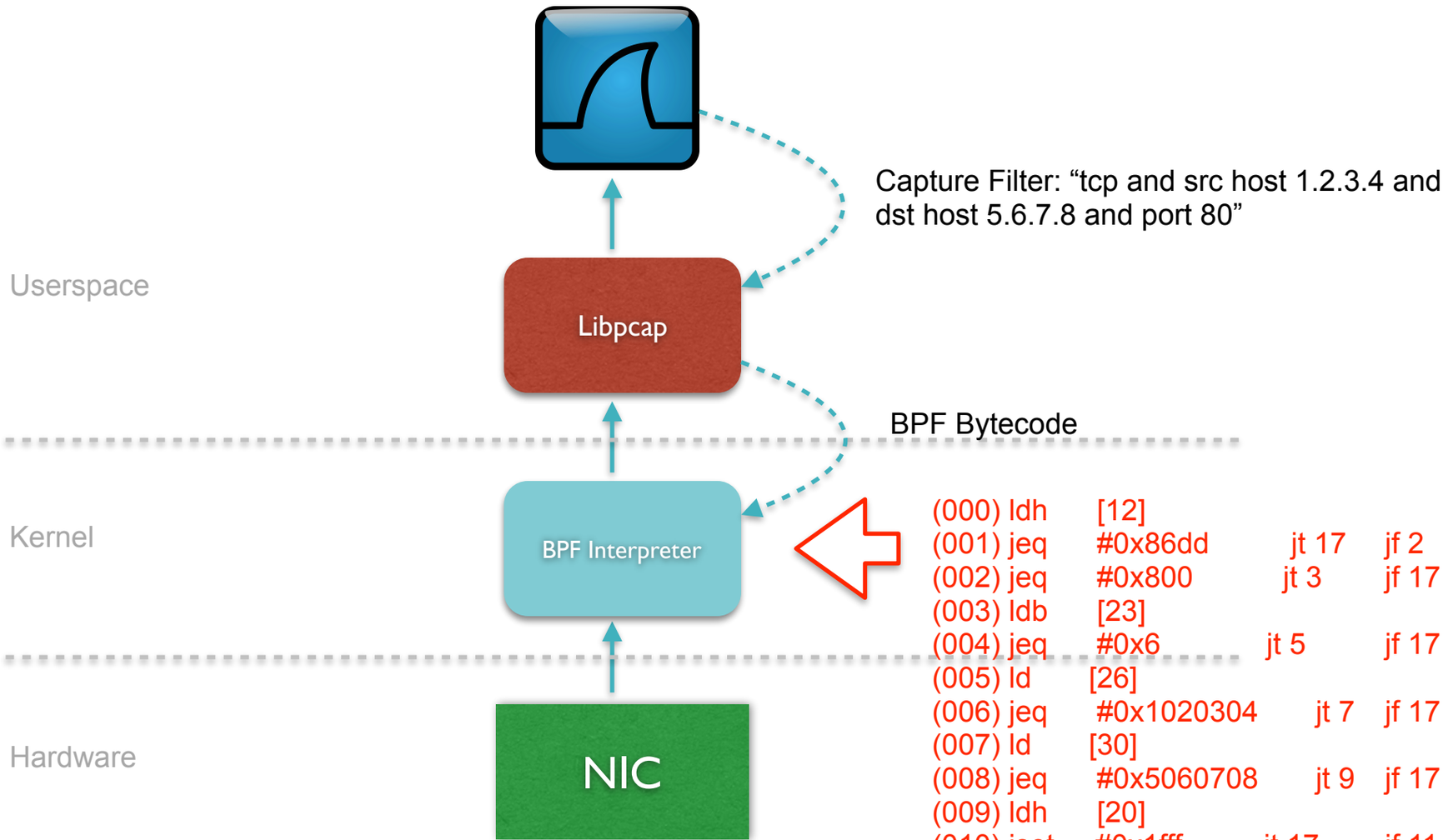
- During troubleshooting we often know in advance what is the traffic we need to analyse: we can filter traffic we're interested in!



ntop

LINUX DAY

Standard BPF

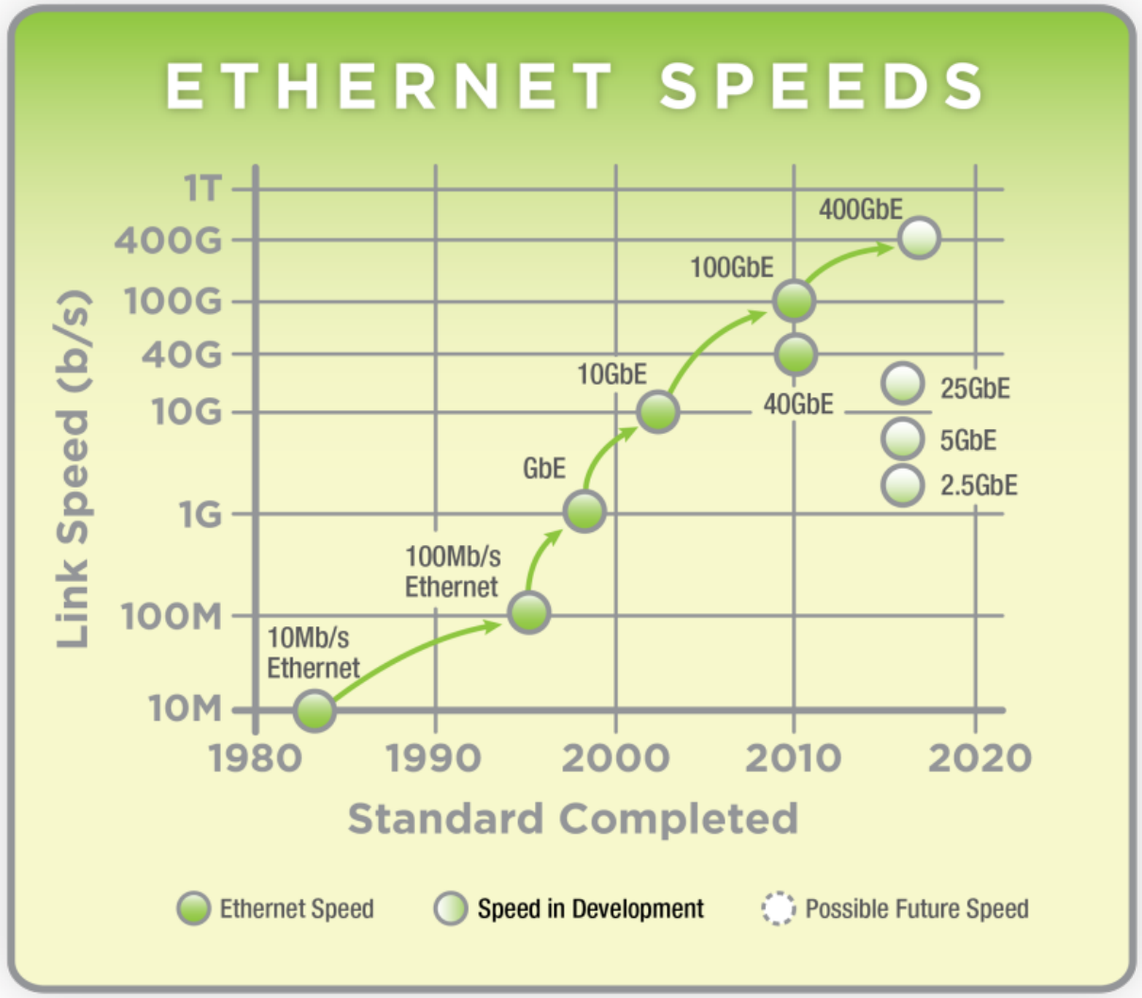


:-\

- Even 10 Gbit is a problem from the packet capture point of view: 14.88 Mpps (1 packet every 67 nsec)

- 100 GbE is already out there

- 400 GbE is coming soon..



- Standard BPF traffic filtering cannot keep up with the ingress rate..

Towards a Light BPF

- Packet filtering can speed up Wireshark but it must be accurate (i.e. no drops during filtering) as drops are not tolerated.

- Is it possible to implement efficient / no-drops packet filtering?

Towards a Light BPF

- Usually people use just a subset of BPF: “tcp and src host 1.2.3.4 and dst host 5.6.7.8 and port 80”

- While BFP has been designed to be very flexible, its flexibility slows down implementations.

- Example (fragments):

```
# tcpdump -i eth1 '((ip[6:2] > 0) and (not ip[6] = 64))'
```

Hw Packet Filtering

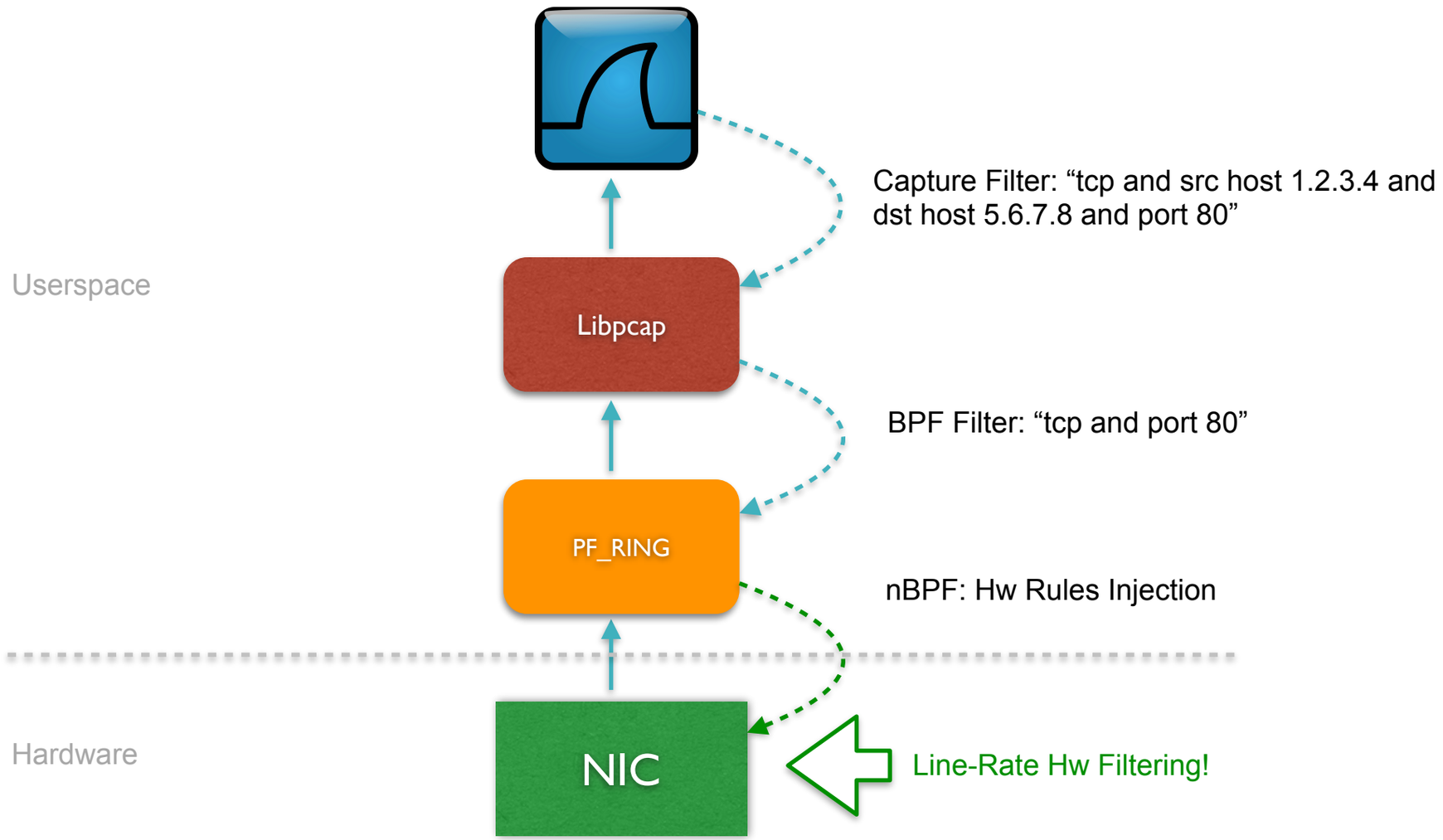
- NTPL (Napatech Packet Language) code for “tcp and src host 1.2.3.4 and dst host 5.6.7.8"

```
DefineMacro(“mUdpSrcPort”,"Data[DynOffset=DynOffUDPFrame;
Offset=0;DataType=ByteStr2]")
DefineMacro(“mUdpDestPort","Data[DynOffset=DynOffUDPFrame;
Offset=2;DataType=ByteStr2]")
DefineMacro(“mTcpSrcPort","Data[DynOffset=DynOffTCPFrame;
Offset=0;DataType=ByteStr2]")
DefineMacro(“mTcpDestPort","Data[DynOffset=DynOffTCPFrame;
Offset=2;DataType=ByteStr2]")
DefineMacro(“mIPv4SrcAddr","Data[DynOffset=DynOffIPv4Frame;
Offset=12;DataType=IPv4Addr]")
DefineMacro(“mIPv4DestAddr","Data[DynOffset=DynOffIPv4Frame;
Offset=16;DataType=IPv4Addr]")
DefineMacro(“mIPv6SrcAddr","Data[DynOffset=DynOffIPv6Frame;
Offset=8;DataType=IPv6Addr]")
DefineMacro(“mIPv6DestAddr”,"Data[DynOffset=DynOffIPv6Frame;
Offset=24;DataType=IPv6Addr]")

Assign[StreamId = 1] = Port == 0 AND (Layer4Protocol == TCP) AND
mIPv4SrcAddr == [1.2.3.4] AND mIPv4DestAddr == [5.6.7.8]
```

Towards a Light BPF

- We want to exploit hardware filters as much as possible while using BPF filters:
  - Preserving the BPF filter syntax (changing it, it's not an option as people are used to)
  - Pushing BPF to hardware

nBPF & Hw filtering



nBPF

- We have created a new user-space BPF engine called nBPF that supports a subset of BPF.

- It has been designed in two layers: filter in hardware what is possible, clean the rest in software.

- nBPF will significantly increment the operational speed and the ability to use Wireshark on a 10/40/100 Gbit NIC without being overwhelmed by ingress traffic as it happens today.

Example - BPF

```
$ tcpdump -i eth1 -d "tcp and src host 1.2.3.4 and dst host 5.6.7.8"

(000) ldh      [12]
(001) jeq      #0x86dd          jt 10 jf 2
(002) jeq      #0x800           jt 3  jf 10
(003) ldb      [23]
(004) jeq      #0x6             jt 5  jf 10
(005) ld       [26]
(006) jeq      #0x1020304       jt 7  jf 10
(007) ld       [30]
(008) jeq      #0x5060708       jt 9  jf 10
(009) ret      #262144
(010) ret      #0
```

} BPF bytecode

Example - nBPF (tree and rules)

```
$ nbpftest -f “tcp and src host 1.2.3.4 and dst host 5.6.7.8"

Dumping BPF Tree
----------------
     Dst Host IP:5.6.7.8
AND
          Src Host IP:1.2.3.4
    AND
          Proto Proto:TCP


Dumping Rules
-------------
[1] [IPv4] [L4 Proto: 6] [1.2.3.4:* -> 5.6.7.8:*]
```

BPF Tree

ACL-like rules

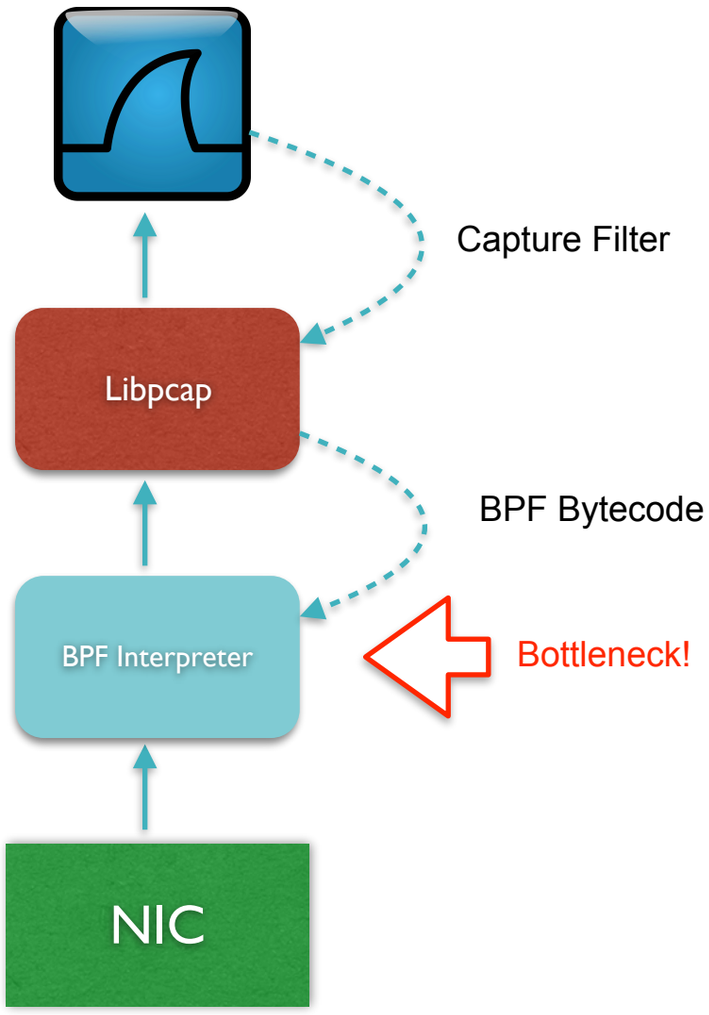
Example - nBPF (Napatech)

- NTPL (Napatech Packet Language) code:

```
$ nbpftest -n -f “tcp and src host 1.2.3.4 and dst host 5.6.7.8"

Napatech Rules
--------------

DefineMacro("mUdpSrcPort","Data[DynOffset=DynOffUDPFrame;Offset=0;DataType=ByteStr2]")
DefineMacro("mUdpDestPort","Data[DynOffset=DynOffUDPFrame;Offset=2;DataType=ByteStr2]")
DefineMacro("mTcpSrcPort","Data[DynOffset=DynOffTCPFrame;Offset=0;DataType=ByteStr2]")
DefineMacro("mTcpDestPort","Data[DynOffset=DynOffTCPFrame;Offset=2;DataType=ByteStr2]")
DefineMacro("mIPv4SrcAddr","Data[DynOffset=DynOffIPv4Frame;Offset=12;DataType=IPv4Addr]")
DefineMacro("mIPv4DestAddr","Data[DynOffset=DynOffIPv4Frame;Offset=16;DataType=IPv4Addr]")
DefineMacro("mIPv6SrcAddr","Data[DynOffset=DynOffIPv6Frame;Offset=8;DataType=IPv6Addr]")
DefineMacro(“mIPv6DestAddr","Data[DynOffset=DynOffIPv6Frame;Offset=24;DataType=IPv6Addr]")

Assign[StreamId = 1] = Port == 0 AND (Layer4Protocol == TCP) AND mIPv4SrcAddr == [1.2.3.4]
AND mIPv4DestAddr == [5.6.7.8]
```

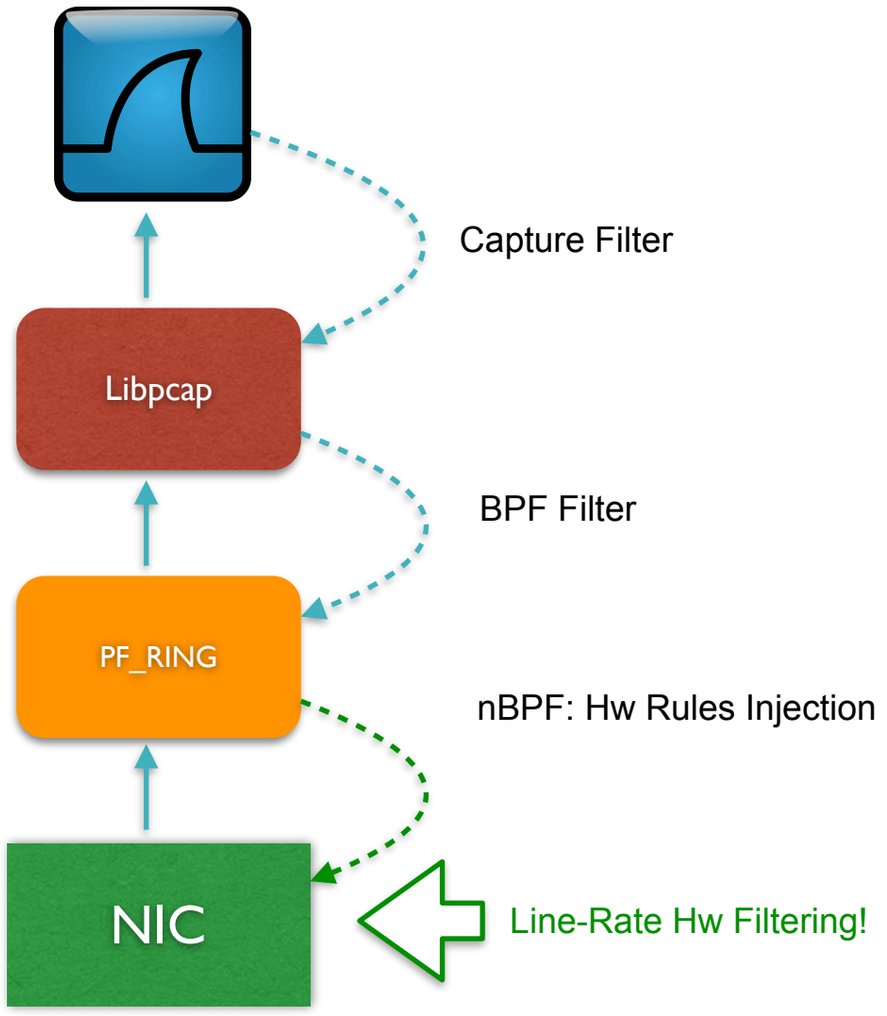
BPF vs nBPF

Standard BPF



100% CPU utilisation,
Packet Loss

Hw nBPF Rules



~0% CPU utilisation

@ 10Gbit, Filtering Out All Traffic

Supported Cards

- Supported cards with hardware filtering:
  - Intel RRC (FM10K)
  - Napatech
  - Exablaze
  - Others soon

Get Started

Source Code (GitHub)

- git clone https://github.com/ntop/PF_RING.git

Packages

- http://packages.ntop.org

Thank you!

ntop