SharkFest '16 Europe

How to Profitably Use Wireshark for Analyzing Large Traces and High-Speed Links

October 18th, 2016

Luca Deri

ntop Founder
deri@ntop.org, #lucaderi

Overview

- Introduction and Motivation
- Multi-10 Gbit Traffic Recording and Indexing
- Improving Wireshark Performance
  - Hardware Packet Filtering
  - Extract Packets From Large (Indexed) pcap Files
- Future Work Items

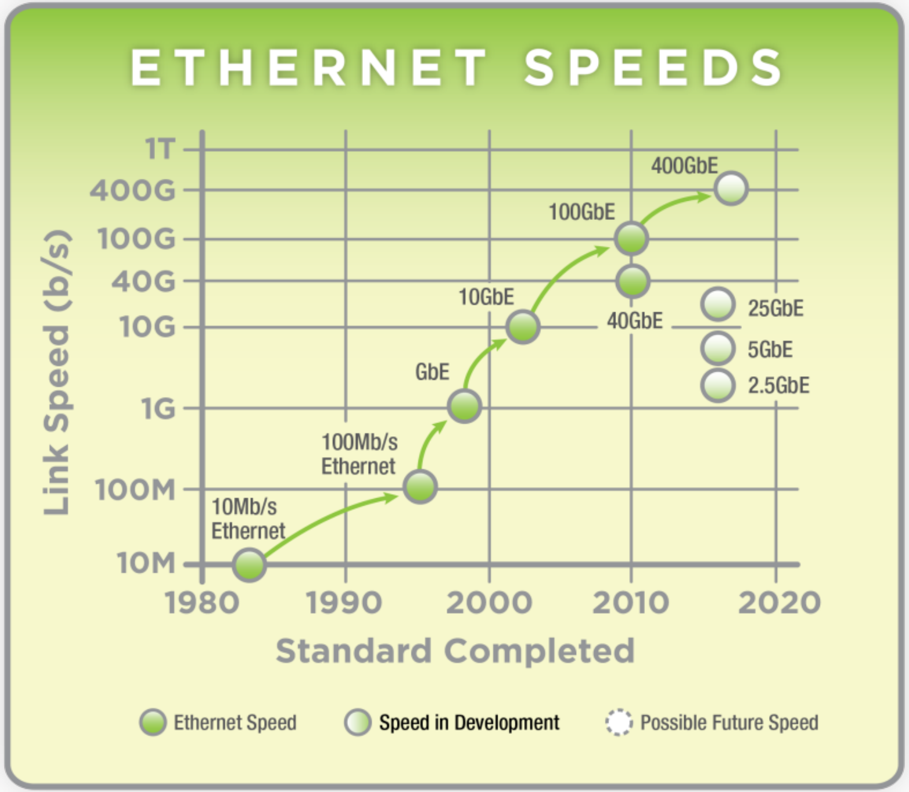
Introduction

- This talk is about creating a comprehensive, high-speed traffic filtering system to be used with Wireshark and other pcap-based applications.
- The goal is to enable Wireshark on 10/40/100 Gbit links or using it to search pcap traces efficiently.
- Software components shown in this talk are either open source or available free of charge (no license required). Commercial applications mentioned are not compulsory for using this work

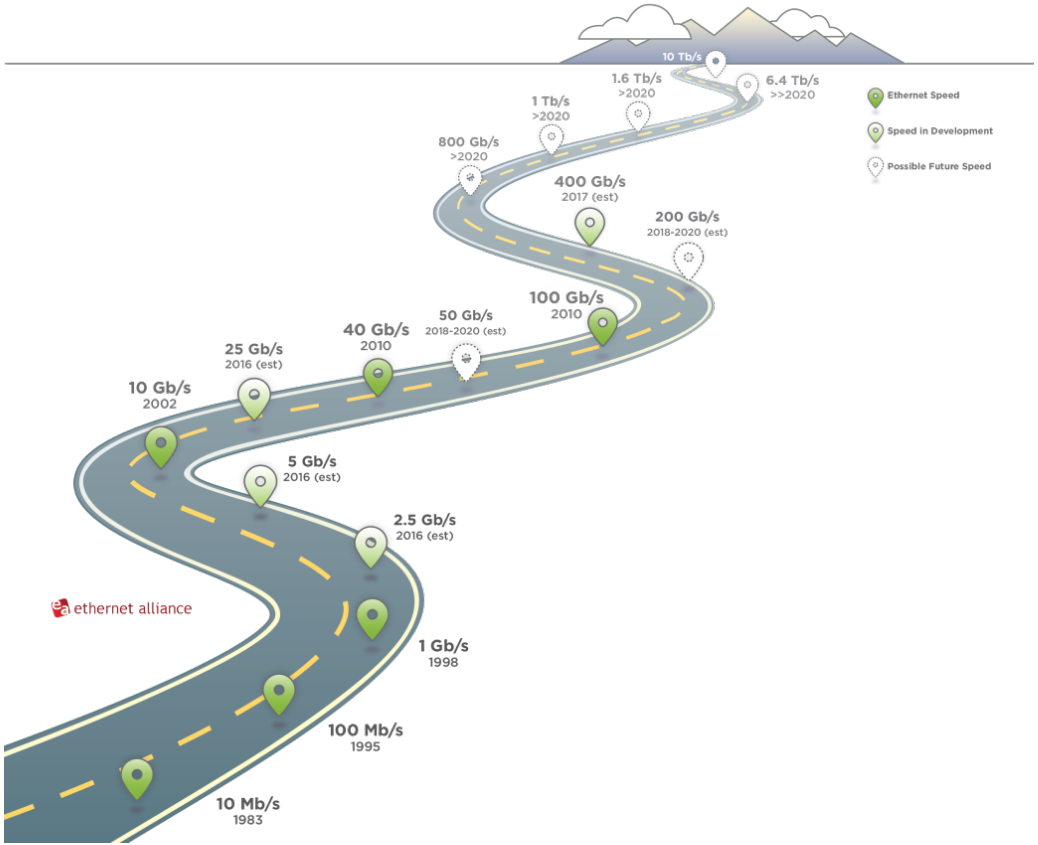
Ethernet Speed is Increasing [1/2]

- Tra 6 new speeds in development: 2.5 GbE, 5 GbE, 25 GbE, 50 GbE, 200 GbE, 400 GbE.
- Cloud transition to 10GbE has passed, pushing fast towards 25G, 50G
- Enterprise servers are still making the transition to 10GbE
- The current 1 GbE will be replaced by 2.5/5 GbE, 10 GbE by 25/50 GbE, 40 GbE by 100 GbE.

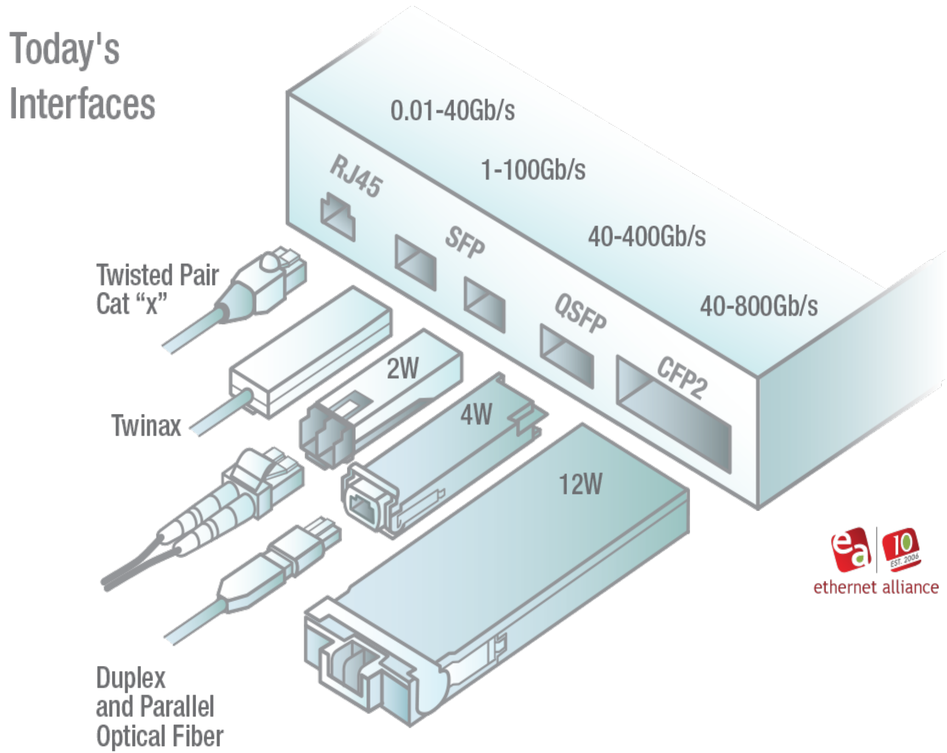
Ethernet Speed Is Increasing [2/2]



Ethernet Roadmap



Ethernet Modules Are Changing Too…



Problem Statement [1/2]

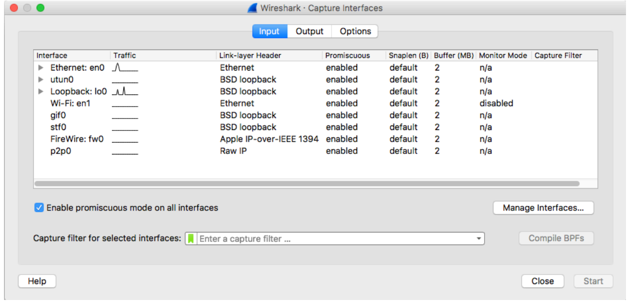
- The ethernet speed will be increasing (practically) in the next few years.
- 10 Gbit is becoming a legacy speed: modern servers already replaced 1G with 10G interfaces.
- But… even 10 Gbit is a problem from the packet capture point of view:
  - 1.25 GB/sec, 14.88 Mpps
  - 5 hours of 10Gbit traffic take ~24TB of disk space

Problem Statement [2/2]

- Wireshark, and most packet monitoring applications, are CPU bound.
- The application performance decreases with the number (and nature) of packets to be analysed.
- Accelerating packet capture can speed-up operations a bit, but over 1 Gbit using Wireshark on live traffic is challenging due to the high number of ingress packets.

Using Wireshark At High-Speed [1/2]

- Wireshark can either analyse packet traces (.pcap) or capture live network traffic.



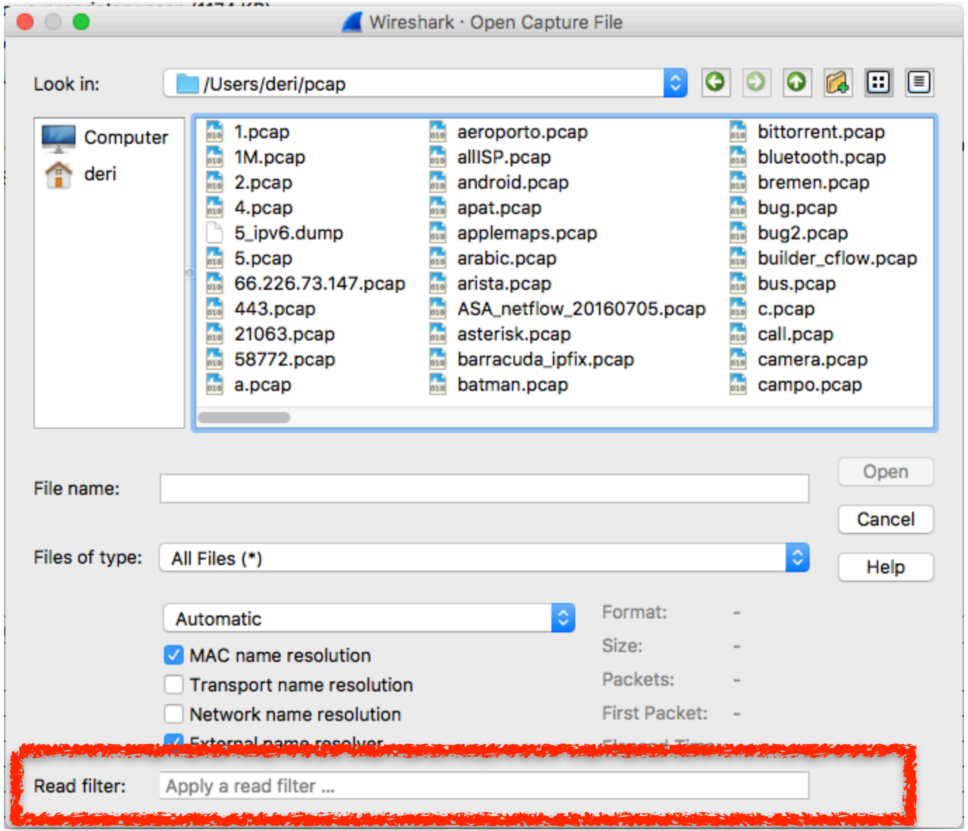
- Live packet capture has hard requirements: at 10Gbit line-rate, there is a packet to process every 67 nsec
  - Too fast for Wireshark.
  - Packet drops make traffic analysis difficult.

Using Wireshark At High-Speed [2/2]

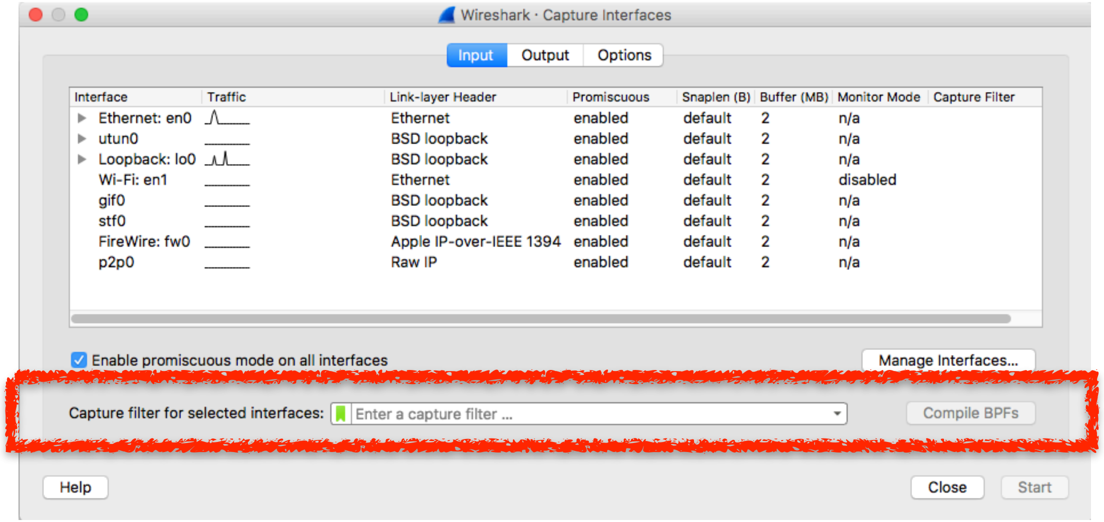
- An option is to capture traffic to disk at line rate and let Wireshark analyse packet dumps.
  - You need to use a packet-to-disk application (e.g. n2disk) to create pcap to disk without dropping anything.
  - Wireshark can then analyse pcaps taking all the time necessary (time is not an issue as it does not cause drops).

However… [1/2]

During troubleshooting we often know in advance what is the traffic we need to analyse with Wireshark (i.e. we can filter traffic we're interested in)



- With live traffic we can also filter traffic in Wireshark , but at high speed it is not very efficient, so packet drops occur making this solution inaccurate and thus useless.



In Summary…

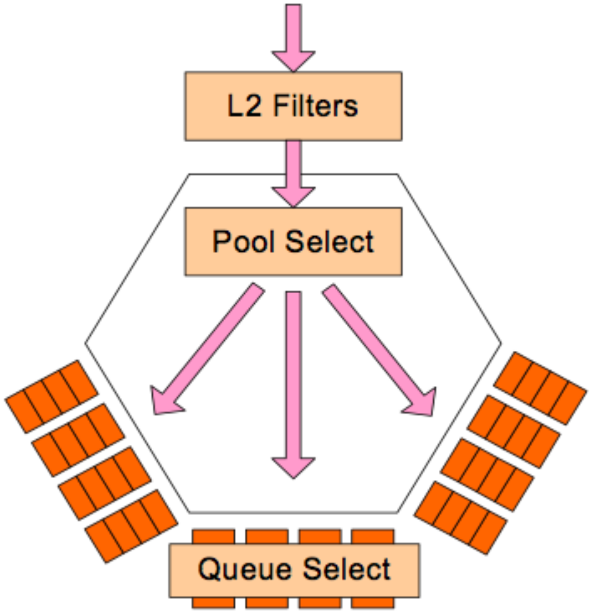
- Packet filtering can speed-up Wireshark but it must be:
  - Accurate (i.e. no drops cause by packet filtering) on live packet capture as drops are not tolerated.
  - Efficient when reading pcap files as users can wait a few seconds but do not usually tolerate waiting time longer than a minute for a packet extraction.

Problem Statement

- Is it possible to implement efficient/no-drops packet filtering during live capture?
- Can we speed-up pcap packet retrieval (e.g. using an index)?
- Can we do this in Wireshark out-of-the-box (i.e. no single line of code change or recompilation will be tolerated) using packaged binaries (e.g. Ubuntu)?
- The answer is YES and we'll explain how in the rest of this talk.

Some Kind of Hw Filters Already Exist [1/3]

- All recent Intel NICs include the Flow Director that can implement flow steering to a virtual RSS queue.
- Thanks to RSS, a physical NIC (e.g. 10 Gbit Intel X520) can be partitioned into n-virtual RX queues (n <= max number of cores)



Some Kind of Hw Filters Already Exist [2/3]

```
ethtool --help:
        ethtool -N|-U|--config-nfc|--config-ntuple DEVNAME    Configure Rx network flow …
        rx-flow-hash tcp4|udp4|ah4|esp4|sctp4|tcp6|udp6|ah6|esp6|sctp6 m|v|t|s|d|f|n|r... |
        flow-type ether|ip4|tcp4|udp4|sctp4|ah4|esp4
            [ src %x:%x:%x:%x:%x:%x [m %x:%x:%x:%x:%x:%x] ]
            [ dst %x:%x:%x:%x:%x:%x [m %x:%x:%x:%x:%x:%x] ]
            [ proto %d [m %x] ]
            [ src-ip %d.%d.%d.%d [m %d.%d.%d.%d] ]
            [ dst-ip %d.%d.%d.%d [m %d.%d.%d.%d] ]
            [ tos %d [m %x] ]
            [ l4proto %d [m %x] ]
            [ src-port %d [m %x] ]
            [ dst-port %d [m %x] ]
            [ spi %d [m %x] ]
            [ vlan-etype %x [m %x] ]
            [ vlan %x [m %x] ]
            [ user-def %x [m %x] ]
            [ action %d ]
            [ loc %d]] |
        delete %d
```

Some Kind of Hw Filters Already Exist [3/3]

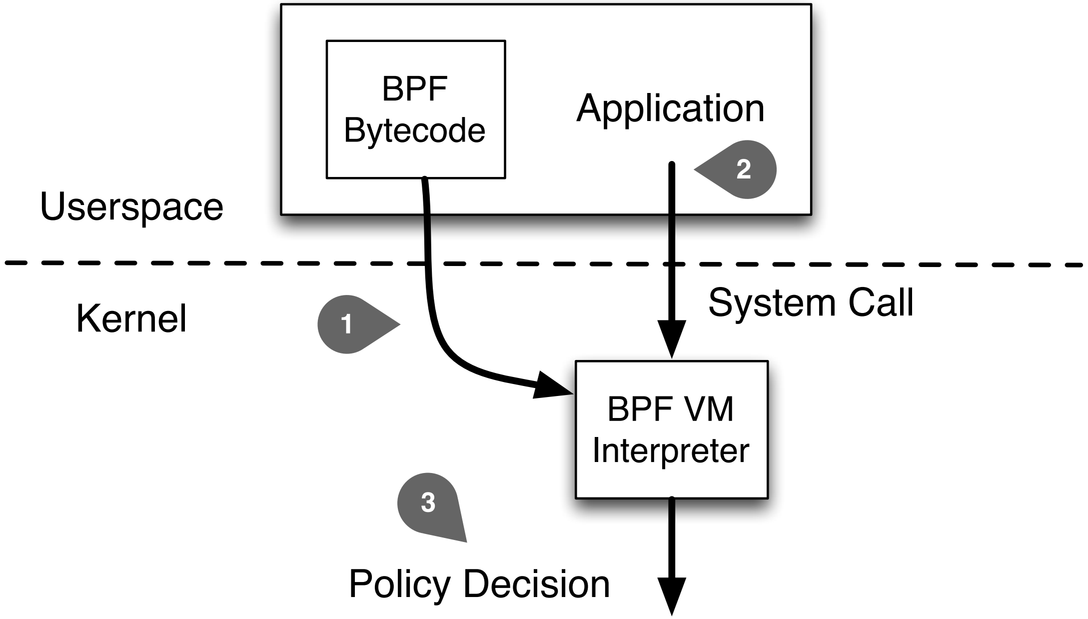
Queue 4

```
# rmmod i40e
# modprobe i40e
# ethtool -X enp6s0f1 weight 1 1 1 1 0 1 1 1
# ethtool -N enp6s0f1 flow-type udp4 dst-port 53 action 4
Added rule with ID 7679
# ethtool -N enp6s0f1 flow-type udp4 src-port 53 action 4
Added rule with ID 7678
# ethtool --show-ntuple enp6s0f1
8 RX rings available
Total 2 rules

Filter: 7678
        Rule Type: UDP over IPv4
        Src IP addr: 0.0.0.0 mask: 255.255.255.255
        Dest IP addr: 0.0.0.0 mask: 255.255.255.255
        TOS: 0x0 mask: 0xff
        Src port: 53 mask: 0xffff
        Dest port: 0 mask: 0xffff
        Action: Direct to queue 4
…
```

Disable Flow Director from queue 4 and set steering rules to queue 4

BPF Intro [1/3]

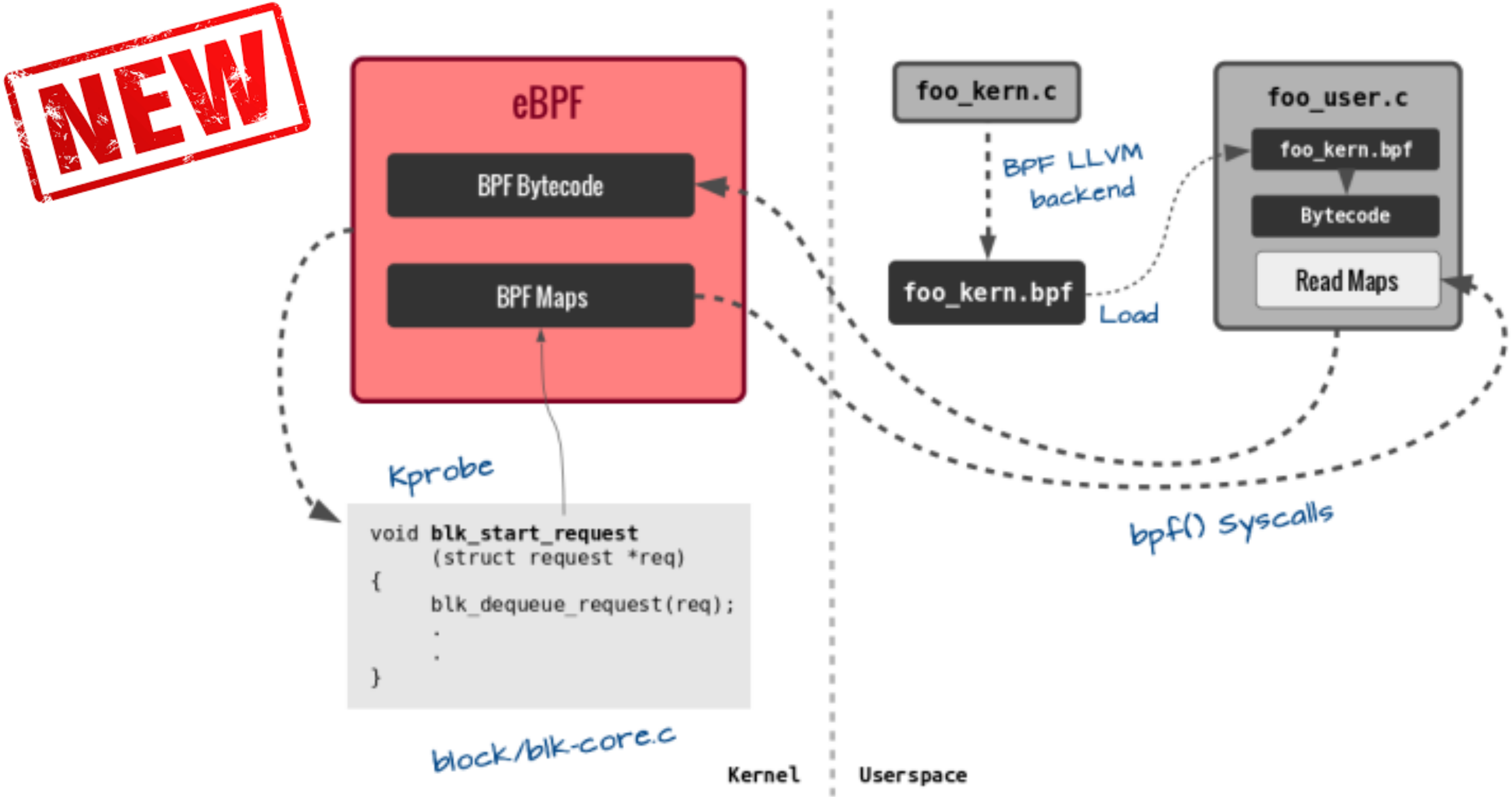
- All libpcap-based applications support BPF that is the de facto filtering mechanism.



BPF Intro [2/3]

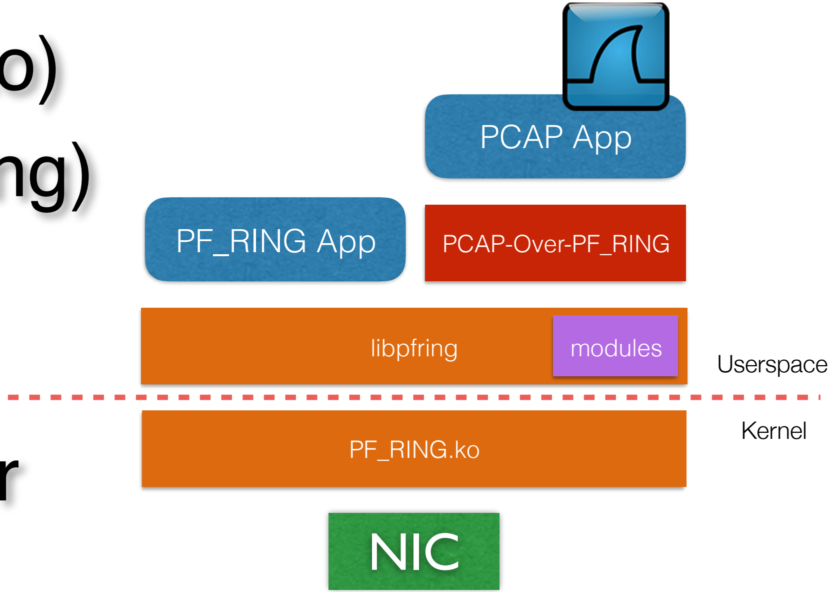
```
# tcpdump -i en0 -d tcp and src host 1.2.3.4 and dst host 5.6.7.8 and port 80

(000) ldh      [12]
(001) jeq      #0x86dd          jt 17   jf 2
(002) jeq      #0x800           jt 3 jf 17
(003) ldb      [23]
(004) jeq      #0x6             jt 5 jf 17
(005) ld       [26]
(006) jeq      #0x1020304       jt 7 jf 17
(007) ld       [30]
(008) jeq      #0x5060708       jt 9 jf 17
(009) ldh      [20]
(010) jset     #0x1fff          jt 17   jf 11
(011) ldxb     4*([14]&0xf)
(012) ldh      [x + 14]
(013) jeq      #0x50            jt 16   jf 14
(014) ldh      [x + 16]
(015) jeq      #0x50            jt 16   jf 17
(016) ret      #262144
(017) ret      #0
```

BPF Intro [3/3]


NEW
eBPF
BPF Bytecode
BPF Maps
Kprobe
void blk_start_request
(struct request *req)
{
blk_dequeue_request(req);
.
.
}
block/blk-core.c
Kernel
Userspace
foo_kern.c
BPF LLVM
backend
foo_kern.bpf
Load
foo_user.c
foo_kern.bpf
Bytecode
Read Maps
bpf() syscalls

Welcome to PF_RING

- PF_RING consists of:
  - Kernel module (pf_ring.ko)
  - Userspace library (libpfring)
  - Userspace modules for multi-vendor support
  - libpcap over PF-RING for legacy applications.
  - Line rate 10/40Gbit RX/TX.



Towards a Light BPF [1/3]

- However very often people use just a subset of it: “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: Match fragments
    # tcpdump -i eth1 '((ip[6:2] > 0) and (not ip[6] = 64))'

Towards a Light BPF [2/3]

- We have realised that:
  - Most people use only a subset of BPF. Popular filters include “proto, IP and port”.
  - Supporting only core BPF filters, makes a BPF engine much faster, lighter, and simpler.
  - We want to exploit hardware filters as much as possible using BPF filters.

Towards a Light BPF [3/3]

- BPF (and pcap) is used both for live traffic capture and pcap file analysis. Seamlessly.
- We must:
  - Preserve the BPF filter syntax (changing it, it's not an option).
  - Push BPF to hardware (live capture) or accelerate it by other means (e.g. index on pcap) on traffic traces.

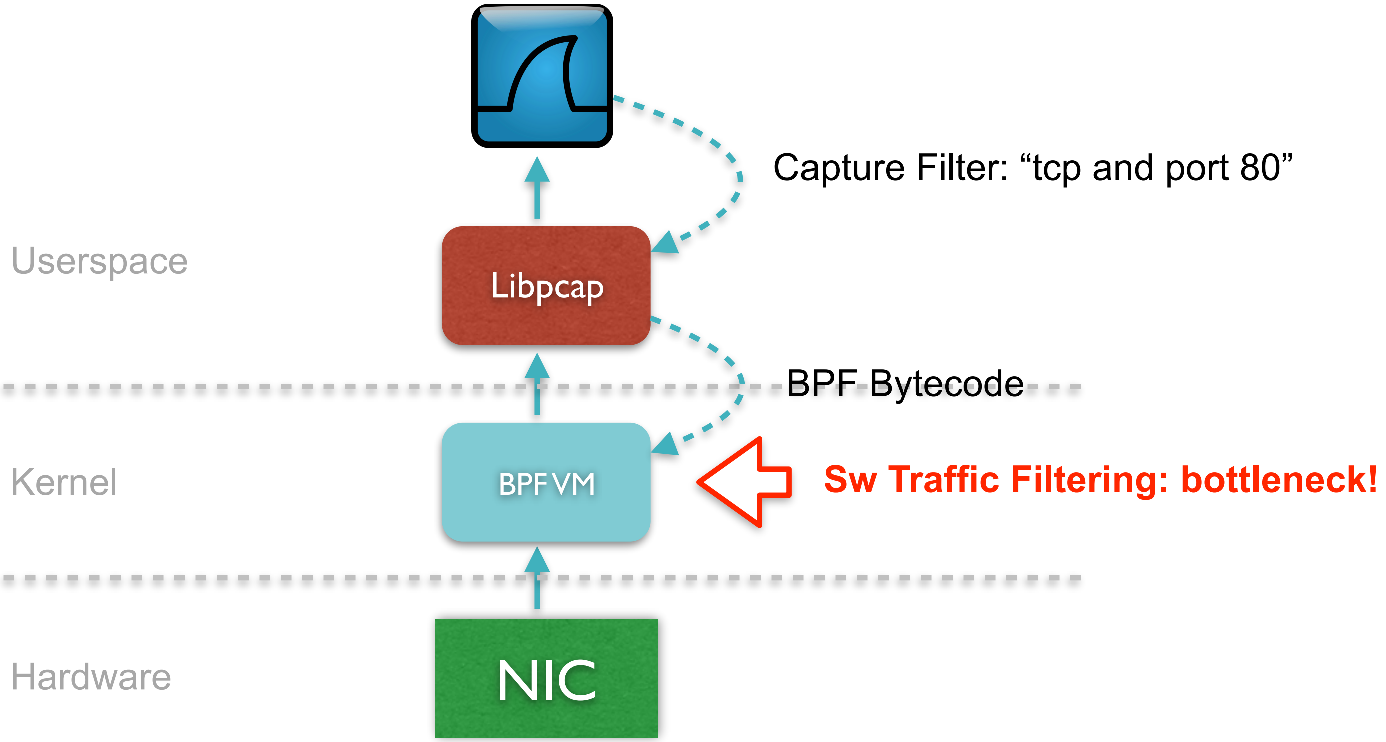
Welcome to nBPF [1/2]

- We have created a new user-space BPF interpreter called nBPF (ntop BPF) that supports a subset of BPF (all popular expressions are supported).
- It has been designed in two layers: filter in hardware what is possible, clean the rest in software if hardware filters can only pre-filter a subset of the traffic.
- When reading from pcaps it must exploit packet indexes to expedite packet extraction.

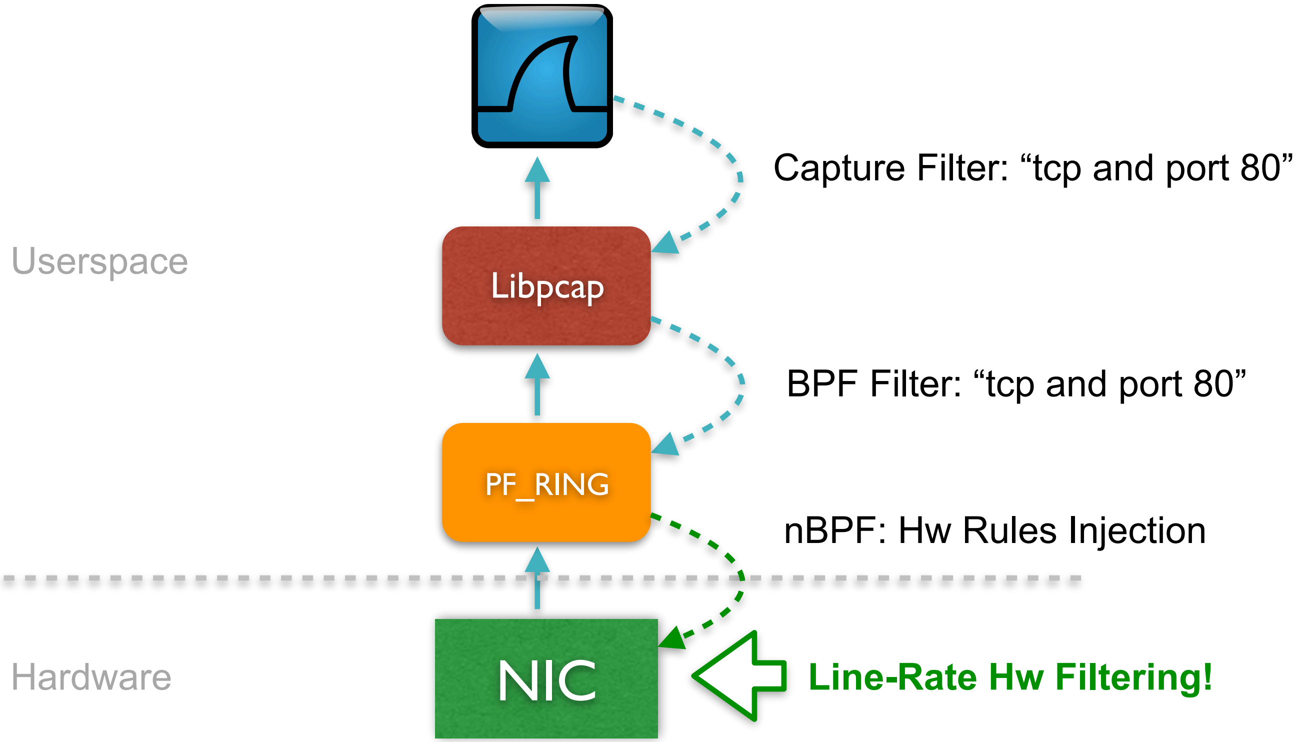
Welcome to nBPF [2/2]

- It must be a *drop-in replacement* for applications that use PF_RING/libpcap: no single line of code has to be changed even in existing applications.
- The only noticeable difference to users with respect to vanilla BPF is in terms of *user experience*:
  - nBPF will significantly increment the operational speed and the ability to use Wireshark on a 10/40/100Gbit NIC in live packet capture without being overwhelmed by ingress traffic as it happens today.

BPF vs nBPF [1/2]



BPF vs nBPF [2/2]



nBPF Expressions [1/3]

- An expression consists of one or more primitives.
- The filter expressions are built by using AND and OR (NOT operation is not permitted).
- Supported Expressions:
  - Protocol: tcp, udp, sctp
  - Direction: src, dst, src or dst, src and dst
  - Type: host, port and protocol

nBPF Expressions [2/3]

Additional constraints for packet capture filters include:

- It is not possible to use more than 1-level nesting using parenthesis.
- It is not possible to use the "or" operator inside parenthesis.
- It is not possible to mix different operators (only 1-level "or" of "and" blocks is allowed).
- It is not possible to combine different directions in the same block using the "and" operator.

nBPF Expressions [3/3]

Valid Filters

- dst host 192.168.0.1
- src port 3000
- ip dst host 192.168.0.1
- src host 192.168.0.1 or dst host 192.168.0.1
- src port 3000 and src host 10.0.0.1 and proto 17
- tcp src port (80 or 443)
- (host 192.168.0.1 and port 3000) or (src host 10.0.0.1 and proto 17)

Unsupported Filters

- src port 3000 and (src host 10.0.0.1 or src host 10.0.0.2)

nBPF and Libpcap

- We have embedded nBFP in PF_RING and thus in libpcap.
- The nBPF parser builds a filter tree memory and then generates a software filtering engine (for post-filtering) and hardware filtering rules.
- In case PF_RING detects that the underlying NIC supports hardware filters, it pushes the filter down to the hardware while enabling the software BPF filter only if necessary.

Evaluating nBPF [1/3]

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

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

Tree-like

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

ACL-like

Evaluating nBPF [2/3]

```
$ tcpdump –i en0 –d "src host 1.2.3.4 and tcp and dst host 5.6.7.8”

(000) ldh      [12]
(001) jeq      #0x800           jt 2 jf 9
(002) ld       [26]
(003) jeq      #0x1020304       jt 4 jf 9
(004) ldb      [23]
(005) jeq      #0x6             jt 6 jf 9
(006) ld       [30]
(007) jeq      #0x5060708       jt 8 jf 9
(008) ret      #262144
(009) ret      #0
```

} VM-like code

Evaluating nBPF [3/3]

```
Napatech Rules
---------------
'DefineMacro("mIPv4SrcAddr","Data[DynOffset=DynOffIPv4Frame;Offset=12;DataType=IPv4Addr]")'
'DefineMacro("mIPv4DestAddr","Data[DynOffset=DynOffIPv4Frame;Offset=16;DataType=IPv4Addr]")'
'Assign[StreamId = 1] = Port == 0 AND (Layer4Protocol == TCP) AND mIPv4SrcAddr == [1.2.3.4]
AND mIPv4DestAddr == [5.6.7.8]'
```

NTPL (Napatech Packet Language)-code

nBPF Supported Network Adapters

Hardware adapters with hardware filters currently supported by nBPF [A-Z]:

- Exablaze
- Intel FM10K
- Napatech

Not Only Live Packet Capture [1/2]

- Live packet capture is not always the best solution for many reasons:
  - Wireshark is not designed to constantly capture traffic.
  - As troubleshooting tool, net admins use it when necessary, not as a permanent monitoring tool.
  - As problems can occur at any time, it is desirable to operate a permanent packet capture tool and filter packets in post-processing.

Not Only Live Packet Capture [2/2]

- Network packet recorders are devices that can continuously write packets to disk.
- The goal is to create a sort of “large buffer” long enough (in time) to allow packets to be filtered/retrieved as long as they are present in the buffer (i.e. before they are overwritten).
- This requires filtering packets on traffic dumps while network traffic is recorded.

Continuous Recording Because… [1/2]

- Network problems can happen at any time.
- Even with real-time monitoring when a issue is detected the packets that created the issue are already gone.
- On-demand recording is not an option as it's not possible to predict and an issue will occur (i.e. your capture will start after the problem has already happened).

Continuous Recording Because… [2/2]

- Continuous recording guarantees that issues are recorded since their inception.
- Capture must be drop-free: the problem can occur during traffic bursts so dropping isn't an option.
- However oldest packet dumps are overwritten as disk space fills up: even with a very large storage system, at some point you will run out of disk space.

Recording is Not Just For Troubleshooting

- Large companies are often protected by a firewall and IDS (Intrusion Detection System): these tools do not keep traffic history but just log security events.
- As in real life, a network packet recorder can help understanding the genesis of the attack (if from the outside) or information leak (if from the inside).
- Thus a continuous packet recorder is mandatory for providing evidence issues and learning how they have originated (and thus repaired).

Packet Shuffling is Not an Option

- Modern network adapters support RSS so that multiple RX queues can be read concurrently to improve packet dumping or filtering performance (i.e. for accelerating software packet filtering).
- RSS has the side effect of shuffling ingress traffic and thus changing the order of network packets
- However shuffling must avoided as shuffling in packet traces won’t help with troubleshooting.

What About Disk Space?

- Packet compression can help depending on traffic type:
  - Most traffic is already compressed (JPEG, MP3)
  - LAN traffic is often uncompressed (SQL, file transfer…)
- The rule of thumb says that you can save ~5% on Internet, and > 50% on LAN traffic.
- RAID is a good option for increasing disk bandwidth:
  - SATA/SAS 10k/15k RPM drives are a good compromise in terms of price/number, SSDs can be fewer/faster but more expensive
  - You need >= 8 SAS drives for 10 Gbit, 32 drives for 40 Gbit.

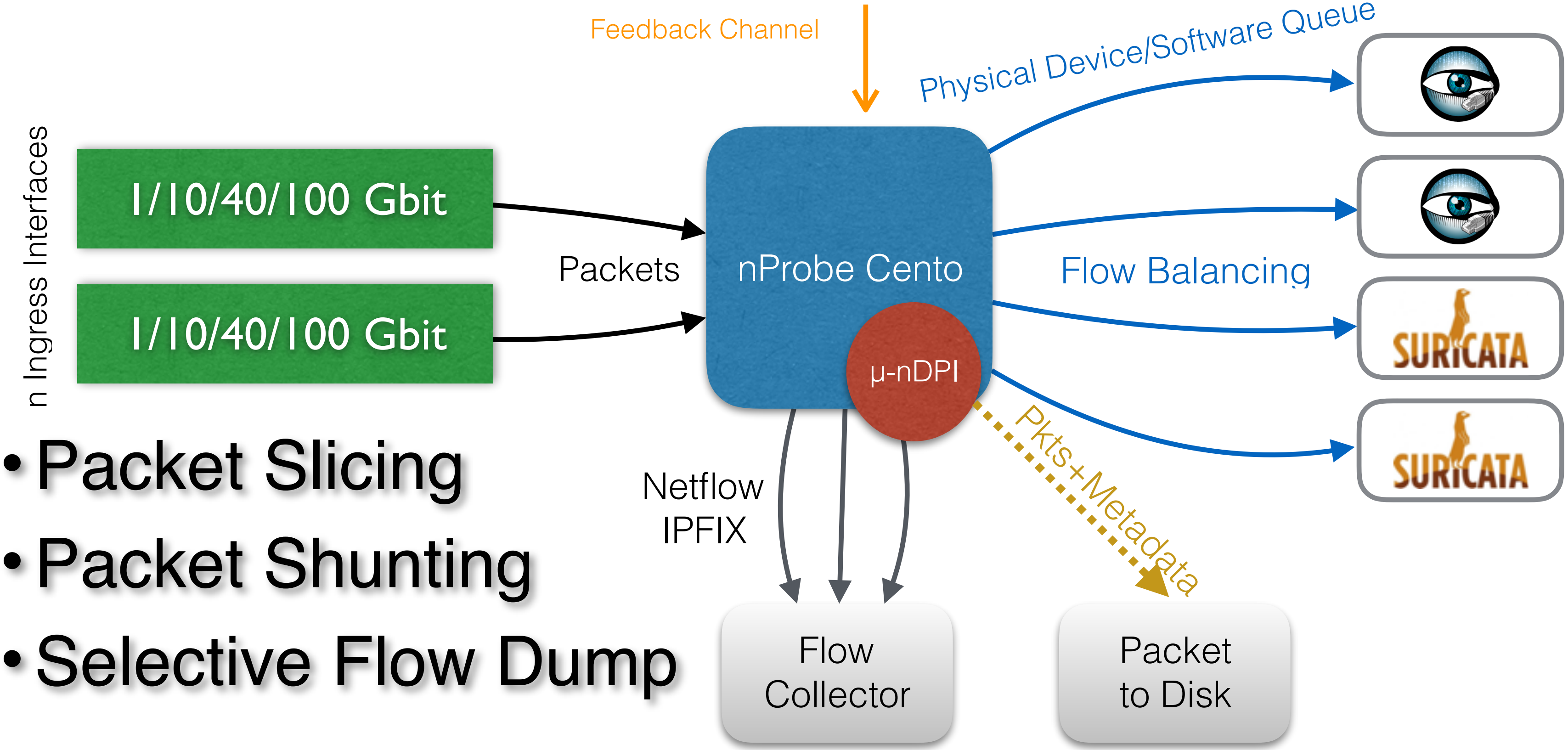
Saving Disk Space Has Many Advantages

- Saving fewer data to disk means less pressure on the disk controller and drives.
- Longer data retention.
- Faster packet search time.
- As recording happens while searching, manipulating smaller files results in fewer I/O and thus less load on the storage system (or if you wish smaller probability of dropping packets due to busy I/O)

Using Packet Filtering to Save Space

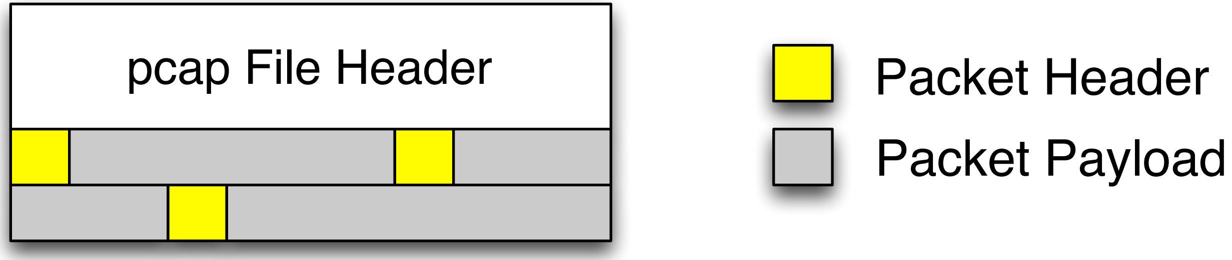
- Filtering can occur during or after capture.
- During capture it allows traffic dumps to be reduced as unwanted traffic is discarded and thus disk space is saved.
  - Caveat: interesting packets can be in the traffic portion you have filtered hence make sure you are NOT filtering meaningful packets.
- Filtering after capture is possible but in this case filtering won’t help you saving disk space.

More Creative Ways To Save Space…



- Packet Slicing
- Packet Shunting
- Selective Flow Dump

Using BPF To Filter Pcap Traces [1/2]



- Pcap files must be read sequentially as the packet header contains no index: in essence when filtering packets the only option is a linear scan.
- BPF can be used, as in live capture, to extract from the pcap only those packets that are meaningful.

Using BPF To Filter Pcap Traces [2/2]

- Pcap packet filtering happens in user space so accelerating packet filtering with hardware filters is not an option (unless you want to inject a pcap to the NIC of course).
- In this case filtering can be accelerated by :
  - Reducing the amount of data read for extraction.
  - Implementing a faster (non VM-based) BPF filtering.

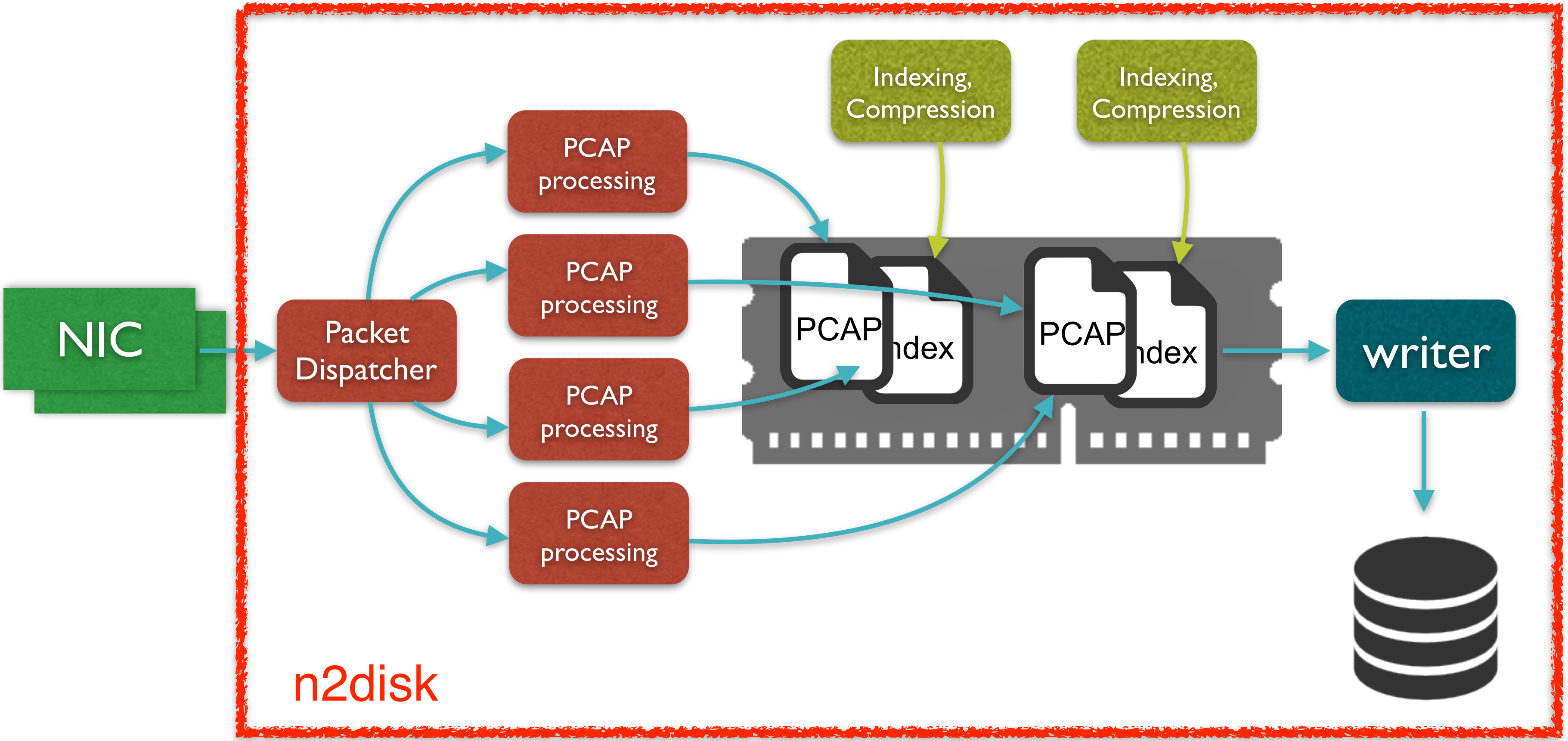
Reducing Pcap Data

- In databases, indexes are used to avoid linear data scan and jump straight to the information we're searching.
- Indexes take space and time and thus they need to be created only on those fields that will be used for searching: VLAN, Mac Address, IPs, Ports and Protocols.
- Unfortunately pcaps have no index…

Indexing while Capturing [1/3]

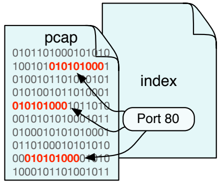
- Got it, I want to create an index on pcaps to speed-up packet filtering. When?
- Post-processing (i.e. and index is computed after the pcap has been saved to disk): not an option as it will put extra pressure on the storage system leading to packet drops.
- During capture: best option but we need to be able to create it at line rate without slowing down packet dump.

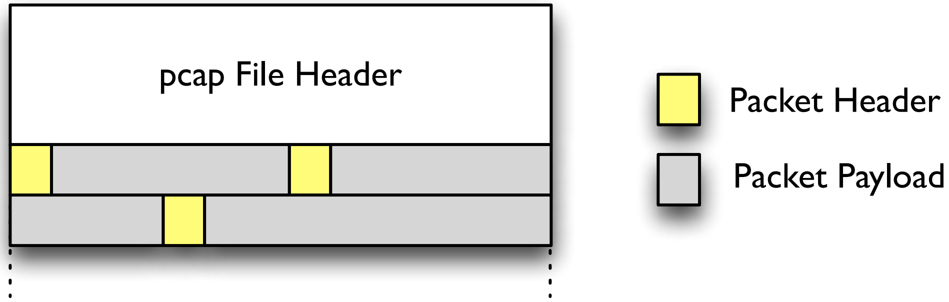
Indexing while Capturing [2/3]

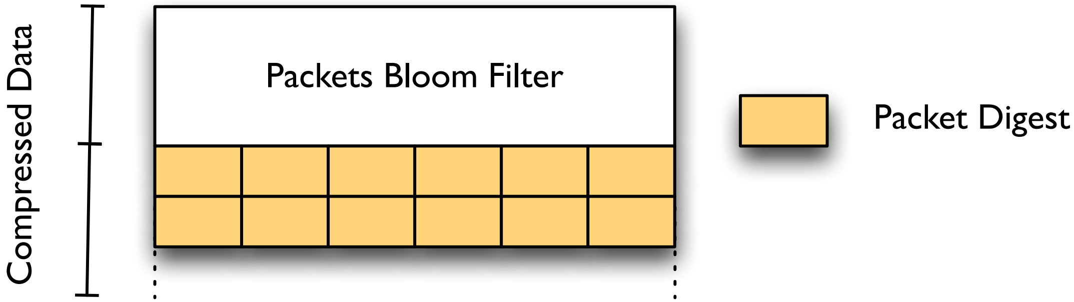


Indexing while Capturing [3/3]

Every pcap file comes with a companion index file

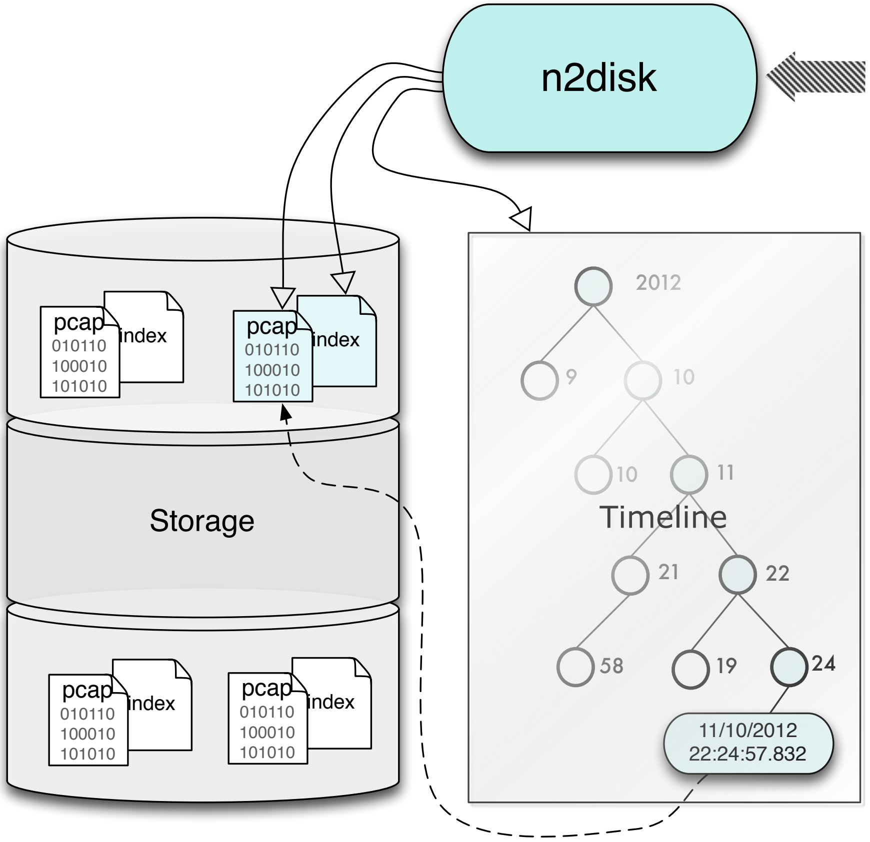






Binding pcaps with Indexes

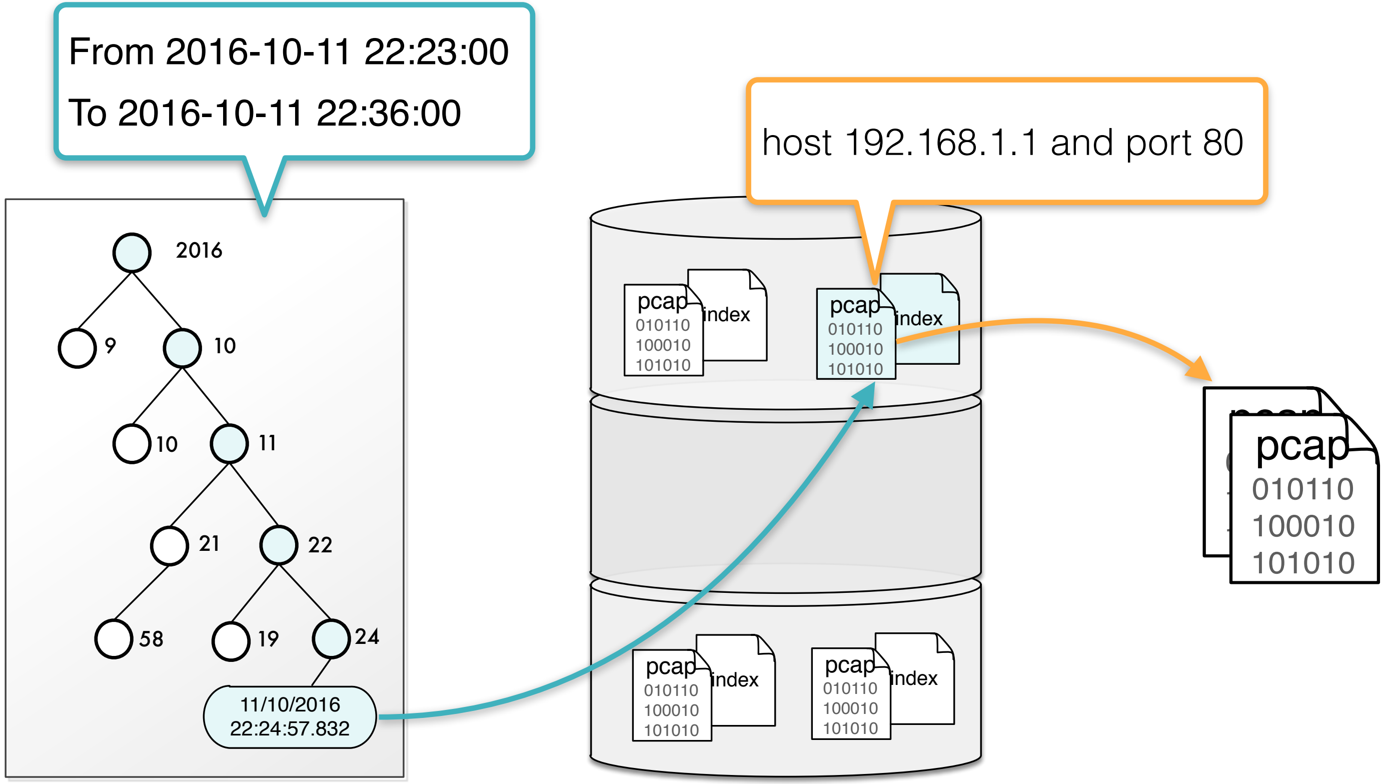
- A time-ordered directory tree maintained by n2disk to enable time-based packet extraction.
- n2disk comes with companion tools for indexing packets in post-processing.



Extracting pcaps Using Indexes [1/2]

- A npcapextract is a companion tool that it can read a tcp file or a dump set (time-ordered pcap files and indexes) created by n2disk.
- The tool produces a new pcap file (or several pcap files according to the specified file limit) with the packets matching the provided filter in BPF-like syntax.

Extracting pcaps Using Indexes [2/2]



Transparently Using Indexes with Wireshark [1/2]

- Running wireshark on an indexed dump set:
  - Accelerates packet retrieval, especially when the extracted packets are a small subset of the whole traffic.
  - It enables data analysis while the extraction (which usually takes time on TBs of data) is still in progress (no need to wait it completes).

Transparently Using Indexes with Wireshark [2/2]

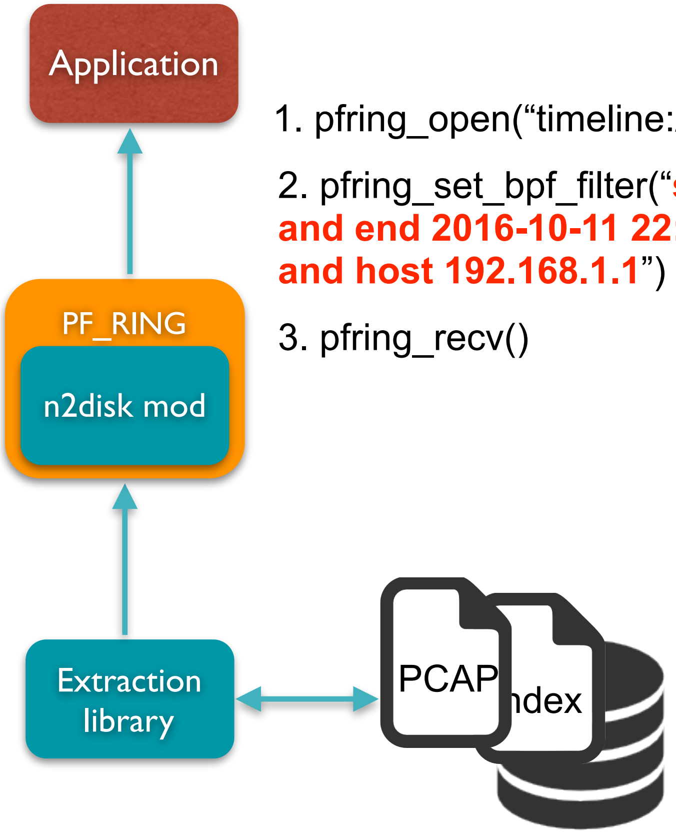
- Sadly Wireshark does not support the n2disk indexes and timeline.
- Solution:
  - Create a virtual device which is visible in Wireshark and represents the dump set.
  - Extend libpcap-over-PF_RING library to extract traffic from n2disk recorded traces (*a-la* npcapextract) when the virtual device is selected

PF_RING Packet Extraction Module [1/2]

- PF_RING is an open source packet processing toolkit developed by ntop.
- The PF_RING packet extraction module can extract traffic using the PF_RING API using nBPF:
  - "timeline:<path>" is used as interface name as it happens with live packet capture.
  - The extraction time interval specified inside the BPF filter, example:

```
start "2016-10-11 22:23:00" and end "2016-10-11 22:36:00"
and host 192.168.1.1
```

PF_RING Packet Extraction Module [2/2]



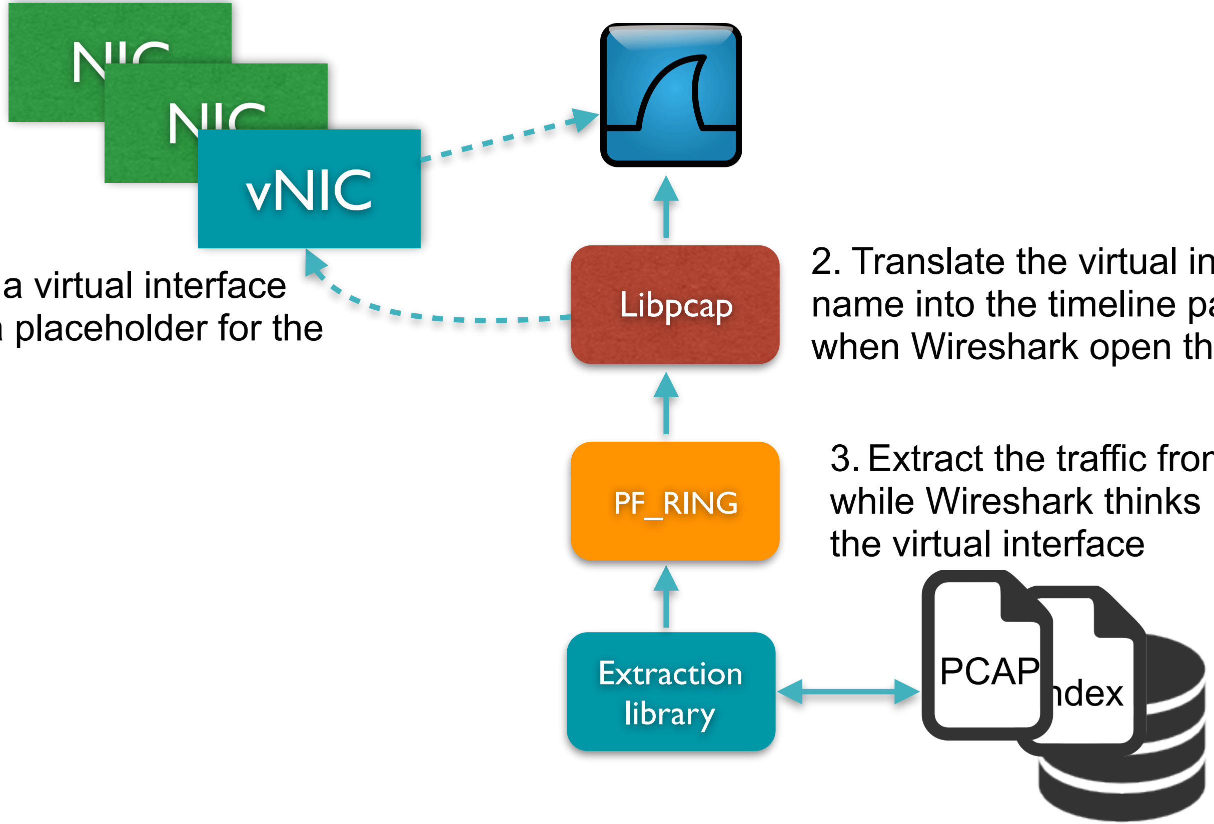
1. pfring_open("timeline:/storage")

2. pfring_set_bpf_filter("**start 2016-10-11 22:23:00 and end 2016-10-11 22:36:00 and host 192.168.1.1**")

3. pfring_recv()

nBPF

Using PF_RING Packet Extraction with Wireshark [1/2]



1. Create a virtual interface which is a placeholder for the dump set

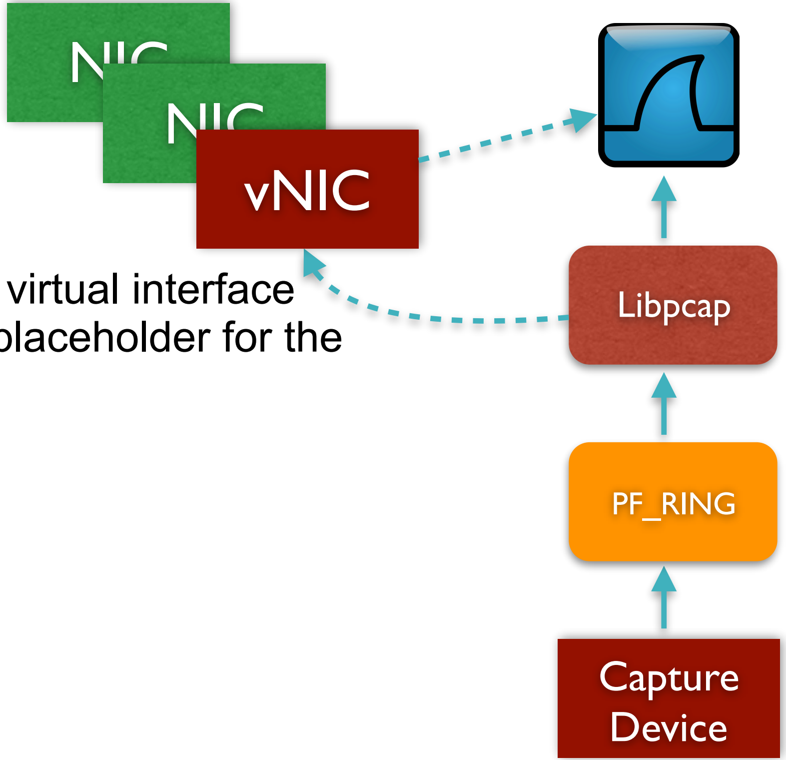
2. Translate the virtual interface name into the timeline path when Wireshark open the interface

3. Extract the traffic from the storage while Wireshark thinks it comes from the virtual interface

Using PF_RING Packet Extraction with Wireshark [2/2]

```
# n2if up -t /storage/n2disk/eth1/timeline -d timeline0
Creating virtual interface timeline0 [timeline: /storage/n2disk/eth1/timeline]
Done
# ifconfig timeline0
timeline0 Link encap:Ethernet  HWaddr ca:35:3b:a8:18:3a
          inet6 addr: fe80::c835:3bff:fea8:183a/64 Scope:Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)
```

Non-standard Network Devices [1/2]



1. Create a virtual interface which is a placeholder for the real device

2. Translate the virtual interface name into the real interface name when Wireshark open the interface

3. Read the traffic from the real interface while Wireshark thinks it comes from the virtual interface

Non-standard Network Devices [2/2]

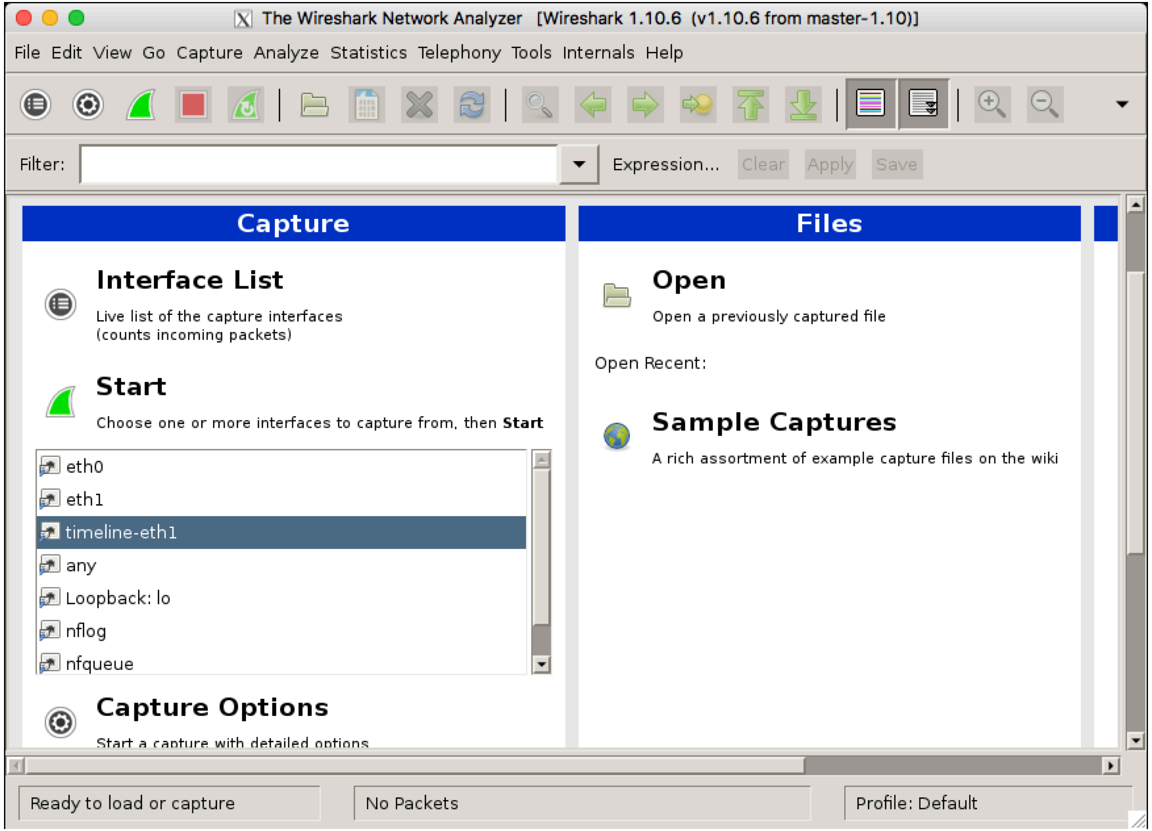
```
# n2if up -i exanic:0 -d exablaze0
Creating virtual interface exablaze0 [associated physical pf_ring interface: exanic:0]
Done
# ifconfig exablaze0
exablaze0 Link encap:Ethernet  HWaddr 6e:cd:b1:59:64:12
          inet6 addr: fe80::6ccd:b1ff:fe59:6412/64 Scope:Link
          UP BROADCAST RUNNING NOARP  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:140 (140.0 B)
```

Demo Time [1/4]

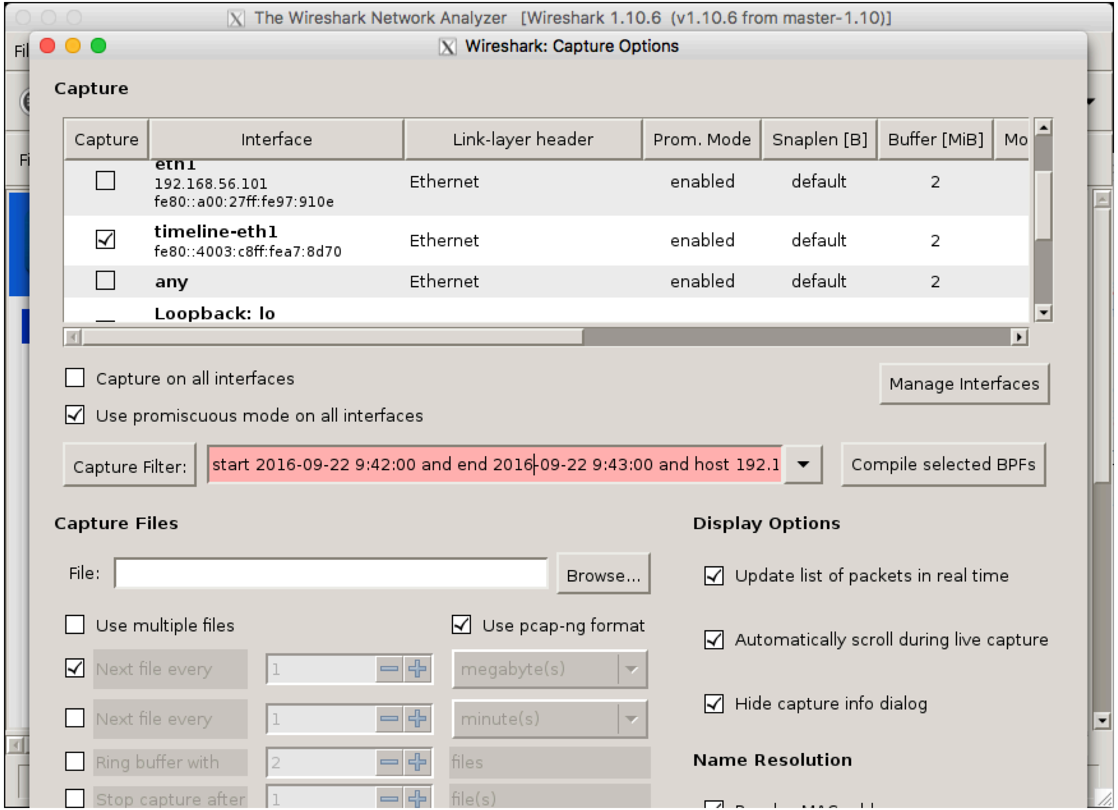
root@nbox: ~ — ssh -Y root@192.168.56.101 — 123×39

```
root@nbox:~# LD_LIBRARY_PATH=/usr/local/lib/ wireshark
wireshark: error while loading shared libraries: libpcap.so.0.8: cannot open shared object file: No such file or directory
root@nbox:~# ln -s /usr/local/lib/libpcap.so /usr/local/lib/libpcap.so.0.8
root@nbox:~# n2if up -t /storage/n2disk/eth1 -d timeline-eth1
Creating virtual interface timeline-eth1 [timeline: /storage/n2disk/eth1]
Done
root@nbox:~# LD_LIBRARY_PATH=/usr/local/lib/ wireshark
```

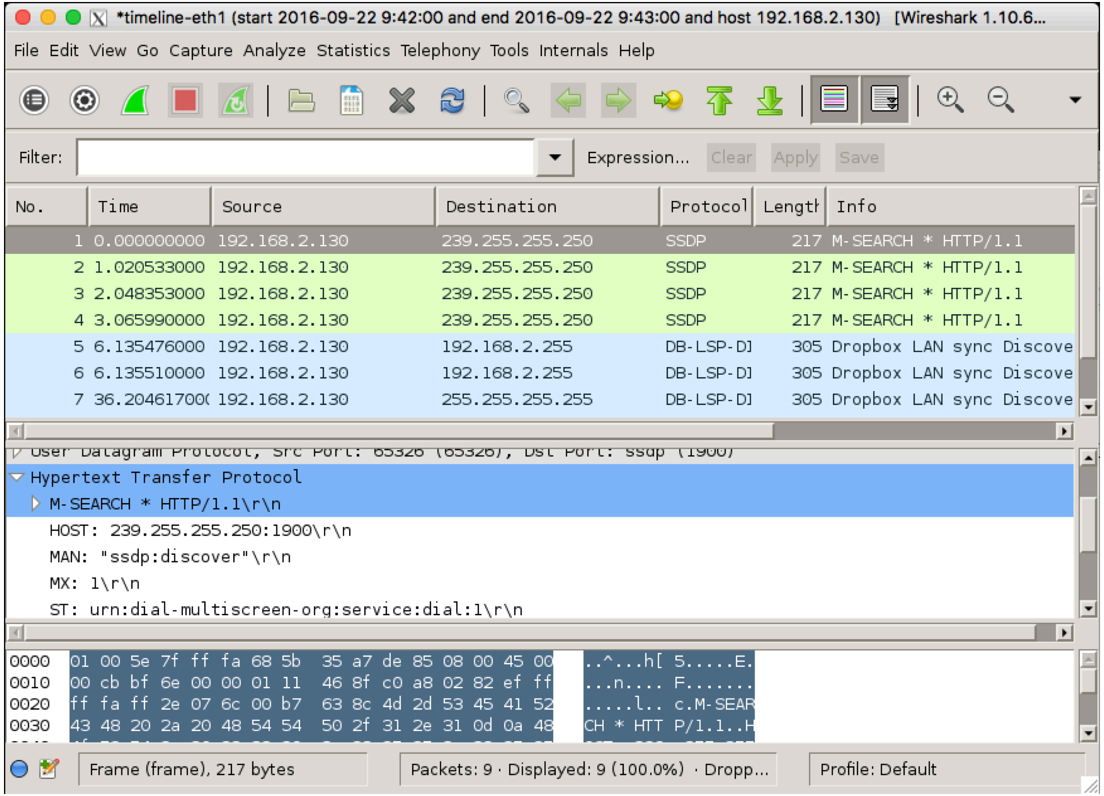
Demo Time [2/4]


The Wireshark Network Analyzer [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Clear
Apply
Save
Capture
Files
Interface List
Live list of the capture interfaces (counts incoming packets)
Start
Choose one or more interfaces to capture from, then Start
eth0
eth1
timeline-eth1
any
Loopback: lo
nflog
nfqueue
Capture Options
Start a capture with detailed options
Open
Open a previously captured file
Open Recent:
Sample Captures
A rich assortment of example capture files on the wiki
Ready to load or capture
No Packets
Profile: Default

Demo Time [3/4]


The Wireshark Network Analyzer [Wireshark 1.10.6 (v1.10.6 from master-1.10)]
Wireshark: Capture Options
Capture
Capture
Interface
Link-layer header
Prom. Mode
Snaplen [B]
Buffer [MiB]
eth1
192.168.56.101
fe80::a00:27ff:fe97:910e
Ethernet
enabled
default
2
timeline-eth1
fe80::4003:c8ff:fea7:8d70
Ethernet
enabled
default
2
any
Ethernet
enabled
default
2
Loopback: lo
Capture on all interfaces
Use promiscuous mode on all interfaces
Manage Interfaces
Capture Filter:
start 2016-09-22 9:42:00 and end 2016-09-22 9:43:00 and host 192.1
Compile selected BPFs
Capture Files
File:
Browse...
Use multiple files
Use pcap-ng format
Next file every
megabyte(s)
Next file every
minute(s)
Ring buffer with
files
Stop capture after
file(s)
Display Options
Update list of packets in real time
Automatically scroll during live capture
Hide capture info dialog
Name Resolution

Demo Time [4/4]


*timeline-eth1 (start 2016-09-22 9:42:00 and end 2016-09-22 9:43:00 and host 192.168.2.130) [Wireshark 1.10.6...
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
1 0.000000000 192.168.2.130 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1
2 1.020533000 192.168.2.130 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1
3 2.048353000 192.168.2.130 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1
4 3.065990000 192.168.2.130 239.255.255.250 SSDP 217 M-SEARCH * HTTP/1.1
5 6.135476000 192.168.2.130 192.168.2.255 DB-LSP-DI 305 Dropbox LAN sync Discove
6 6.135510000 192.168.2.130 192.168.2.255 DB-LSP-DI 305 Dropbox LAN sync Discove
7 36.20461700(192.168.2.130 255.255.255.255 DB-LSP-DI 305 Dropbox LAN sync Discove
Hypertext Transfer Protocol
M-SEARCH * HTTP/1.1\r\n
HOST: 239.255.255.250:1900\r\n
MAN: "ssdp:discover"\r\n
MX: 1\r\n
ST: urn:dial-multiscreen-org:service:dial:1\r\n
0000 01 00 5e 7f ff fa 68 5b 35 a7 de 85 08 00 45 00 ..^...h[5.....E.
0010 00 cb bf 6e 00 00 01 11 46 8f c0 a8 02 82 ef ff ...n.... F.......
0020 ff fa ff 2e 07 6c 00 b7 63 8c 4d 2d 53 45 41 52 .....l.. c.M-SEAR
0030 43 48 20 2a 20 48 54 54 50 2f 31 2e 31 0d 0a 48 CH * HTT P/1.1..H
Frame (frame), 217 bytes
Packets: 9 · Displayed: 9 (100.0%) · Dropp...
Profile: Default

Tools Availability [1/2]

- PF_RING and nBPF: https://github.com/ntop/PF_RING
- n2disk (available at http://packages.ntop.org) is a commercial tool for line-rate multi-10 Gbit packet capture, that we make it available for free to no-profit, research, education.
- n2disk companion tools (index and packet extract) are free of charge.

Tools Availability [2/2]

- n2disk indexing and pcap extraction tools (part of the n2disk package) do not require a license.
- This means that if you don't want to use n2disk to capture traffic, you can use:
  - Wireshark, tshark or tcpdump to create pcap dumps.
  - n2disk indexing tools for building pcap indexes.
  - nBPF/PF_RING for packet retrieval.

Live Demo at Sharkfest

- Get the USB stick from the ntop team
- Copy the `ntop-meeting-hands-on` folder into your PC
- Enter into `ntop-meeting-hands-on`
- Run the VM:
  - `vagrant box add ntop-box ntop-hands-on.box`
  - `vagrant up`
- SSH into the VM:
  - `vagrant ssh`

VAGRANT

Thank you!