Ringmap Packet Capturing Stack

Alexander Fiveg
afiveg@freebsd.org

October 8, 2010

Outline

Motivation

Problems arising during packet capturing

- High bit rates and packet rates
  $\Rightarrow$ **high CPU load and packet loss**
- "expensive" operations in terms of CPU cycles:

Problems arising during packet capturing

- High bit rates and packet rates
  $\Rightarrow$ **high CPU load and packet loss**
- "expensive" operations in terms of CPU cycles:
  - Memory allocations
  - Data copy operations
  - System calls
  - etc. . .

Disadvantages of Standard Packet Capturing Stack

- Memory allocations
  - For each new received packet a new *mbuf* is allocated

(*) Using Zero-Copy BPF eliminates the last copy and system call

Disadvantages of Standard Packet Capturing Stack

- Memory allocations
  - For each new received packet a new *mbuf* is allocated
- Too many packet copies
  - DMA: $Controller \Rightarrow RAM$
  - BPF: $RAM \Rightarrow RAM(BPF\ Buffer)$
  - read(2): $RAM \Rightarrow RAM(Userspace\ Buffer)(*)$

(*) Using Zero-Copy BPF eliminates the last copy and system call

Disadvantages of Standard Packet Capturing Stack

- Memory allocations
  - For each new received packet a new *mbuf* is allocated
- Too many packet copies
  - DMA: $Controller \Rightarrow RAM$
  - BPF: $RAM \Rightarrow RAM(BPF\ Buffer)$
  - read(2): $RAM \Rightarrow RAM(Userspace\ Buffer)(*)$
- System calls
  - User-space application receives the packets using read(2)*(*)*
  - Saving packets to the hard disk

(*) Using Zero-Copy BPF eliminates the last copy and system call

Approach

Approach

- Using **ring** buffers
  - for eliminating the memory allocations
    - reusing packet buffers: no new allocations

Approach

- Using **ring** buffers
  - for eliminating the memory allocations
    - reusing packet buffers: no new allocations
- Using shared memory buffers (*memory* **mapping**)
  - Eliminating the packet copy operations
    - mapping DMA buffers into the user-space

Approach

- Using **ring** buffers
  - for eliminating the memory allocations
    - reusing packet buffers: no new allocations
- Using shared memory buffers (*memory* **mapping**)
  - Eliminating the packet copy operations
    - mapping DMA buffers into the user-space

$\Rightarrow$ Project Name: *ring* + *mapping* = **ringmap**

Approach 2

Supported Hardware:

- The following *Intel GbE Controllers:*
  - *8254x, 8257x, 8259x*

Modified libpcap

- Libpcap is adapted to *ringmap*
  - libpcap-apps don't require modifications
    - tcpdump, wireshark, etc. . .

Achieved goals

Achieved goals 1

- **Portability**
  - Ringmap is easily portable to other ethernet controllers. Only hardware dependent part of code require modifications.
  - The generic driver and libpcap contain a few hooks for calling *ringmap*-functions.
- **Packet Filtering**
  - Packet filtering can be accomplished using both *libpcap*- and kernel-BPF.

Achieved goals 2

- **Multithreaded Capturing**
  - Multiple applications can capture the packets from the same interface.
- **Partly ported to the 10GbE controller**
  - Currently only one queue is used while capturing. The work on supporting multiqueue is in progress.

Achieved goals 3

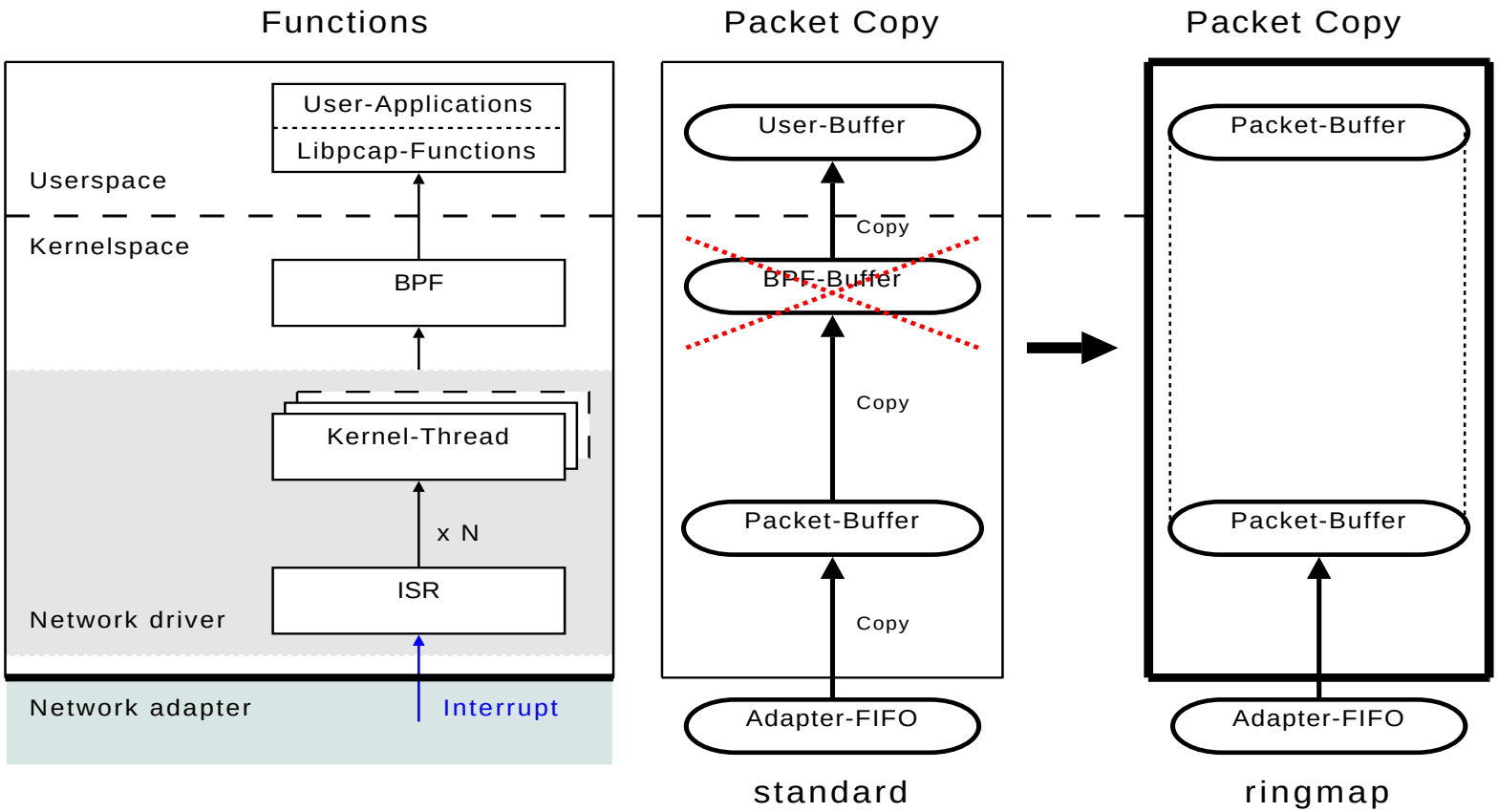
- **Enhanced Capturing-Performance**
  - very low system load (below 12%) and very low packet loss (below 0.02%)
- **Usability of implemented software**
  - Libpcap-applications don't require modification in order to run with the *ringmap*

Future works

- Benchmarking: **Zero-Copy BPF** vs. **ringmap**
- Support for hardware time stamping
- Writing the packets to the disc from within the kernel
- 10-GbE-Packet-Capturing:
  - Multiqueue support
  - Support for hardware packet filtering
- Extending **ringmap** for packet transmission

Thank you! Questions?

Overview


Functions
Packet Copy
Packet Copy
User-Applications
Libpcap-Functions
Userspace
Kernelspace
BPF
Kernel-Thread
x N
ISR
Network driver
Network adapter
Interrupt
User-Buffer
Copy
BPF-Buffer
Copy
Packet-Buffer
Copy
Adapter-FIFO
standard
Packet-Buffer
Packet-Buffer
Adapter-FIFO
ringmap