Ringmap Capturing Stack for High Performance Packet Capturing in FreeBSD

Alexander Fiveg
afiveg@freebsd.org

September 26, 2010

Outline

1. Motivation
2. Background
3. Solution
4. Performance evaluation
5. Summary

Motivation

Problem

Problems arising during packet capturing:

- High bit rates and packet rates
  - $\Rightarrow$ high CPU load and packet loss

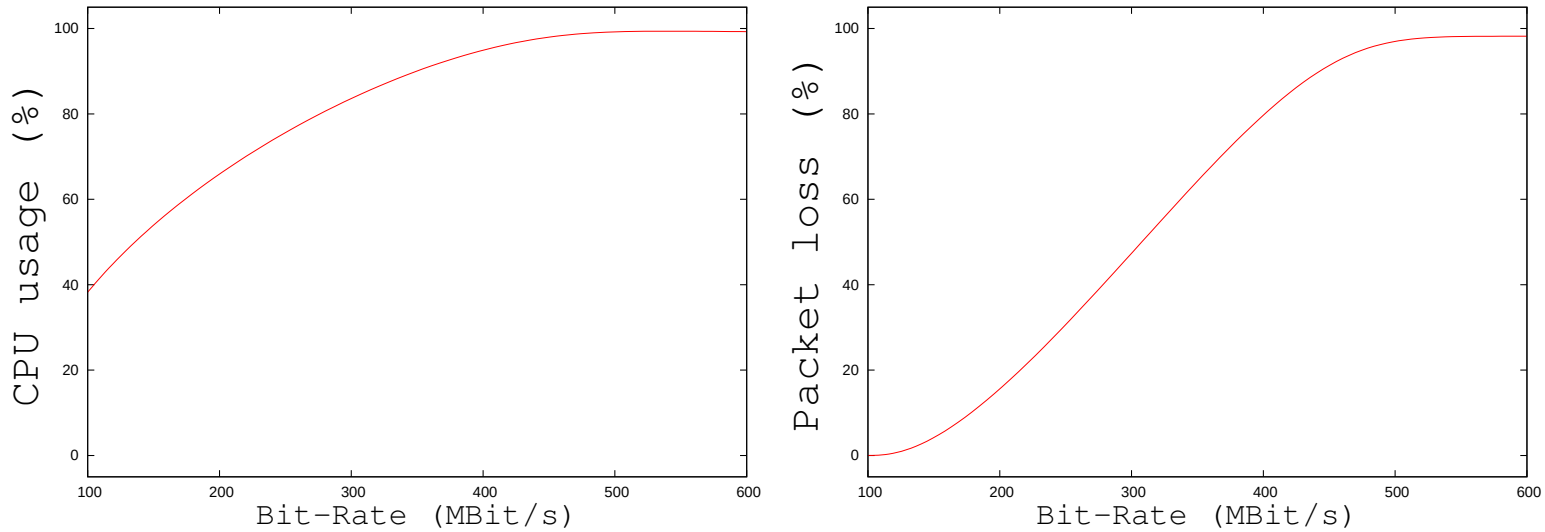


Figure: Capturing 64-Bytes packets. (Bezier curves)

The reason of the Problems

Inefficiency of standard capturing software

- To many "expensive" operations in terms of CPU cycles:
  - System calls
  - Packet copy operations
  - Memory allocations
  - etc. . .

Goal of the project

Increase the capturing performance in FreeBSD

- Design and implementation the new capturing software to minimize the **packet loss** and **CPU load** during capturing.

Supported Hardware:

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

Approach

- Eliminating the packet copy operations
  - by using shared memory buffers (*memory* **mapping**)
- Eliminating the memory allocations
  - by using **ring** buffers

Approach

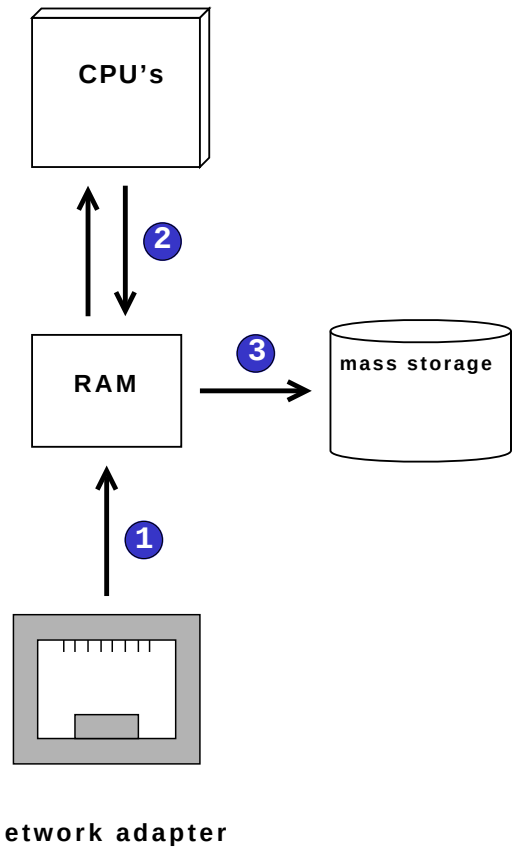
- Eliminating the packet copy operations
  - by using shared memory buffers (*memory* **mapping**)
- Eliminating the memory allocations
  - by using **ring** buffers

$\Rightarrow$ *ring* + *mapping* = **ringmap**

Background

Packet Capturing: Hardware View

1. **Receiving** network packets
   - receive at network adapter
   - DMA transfer in RAM
2. **Filtering** the received packets
   - *Berkeley Packet Filter*
3. **Storing** to the hard disk
   - using a system call



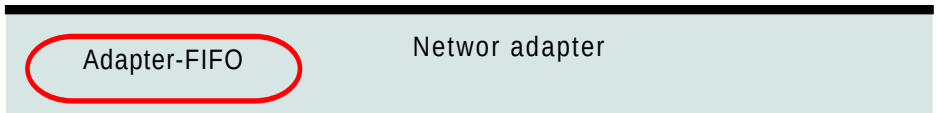
FreeBSD Packet Capturing Stack

- **Network driver**
  - Receiving packets
- **Berkley Packet Filter** (BPF)
  - Filtering packets
- **User-space application**
  - Accessing received packets
  - Initiates:
    - Storing packets to hard drive
    - Output packets information to the terminal

How does standard packet capturing works

FreeBSD Standard Packet Capturing Stack

- Packet is received and saved in Adapter-FIFO



FreeBSD Standard Packet Capturing Stack

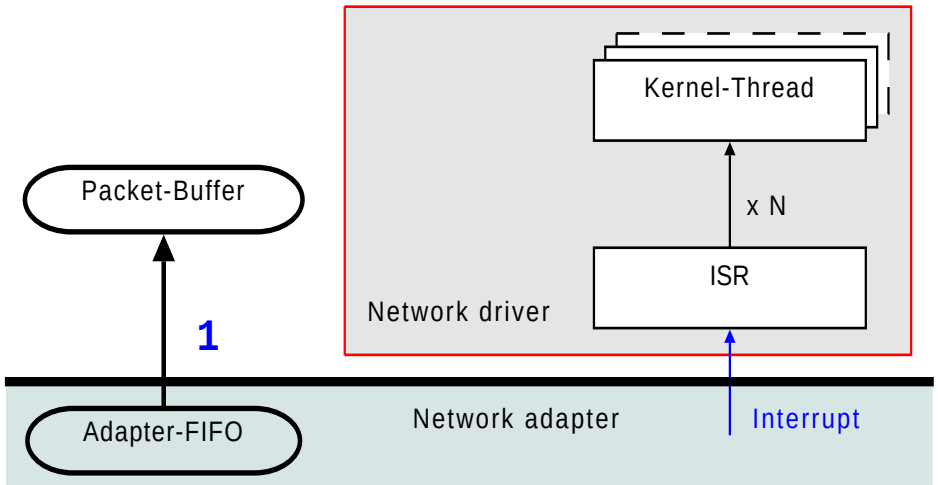
- DMA-Transfer
- Packet is received and saved in Adapter-FIFO

FreeBSD Standard Packet Capturing Stack

- Interrupt
- DMA-Transfer
- Packet is received and saved in Adapter-FIFO

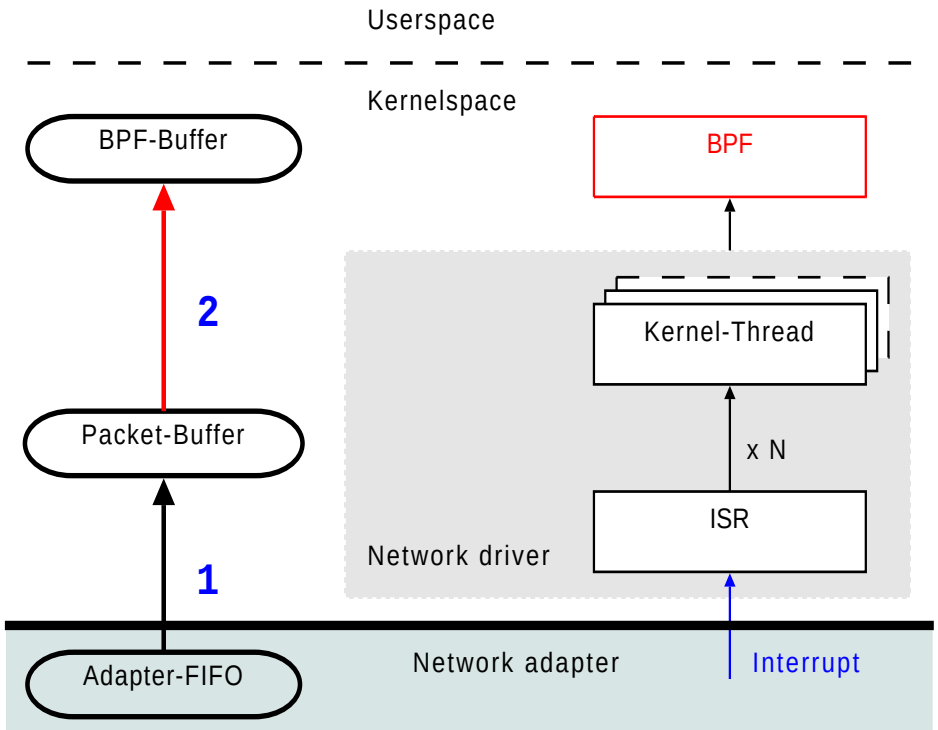
FreeBSD Standard Packet Capturing Stack

- Interrupt Service Routine
- Interrupt
- DMA-Transfer
- Packet is received and saved in Adapter-FIFO



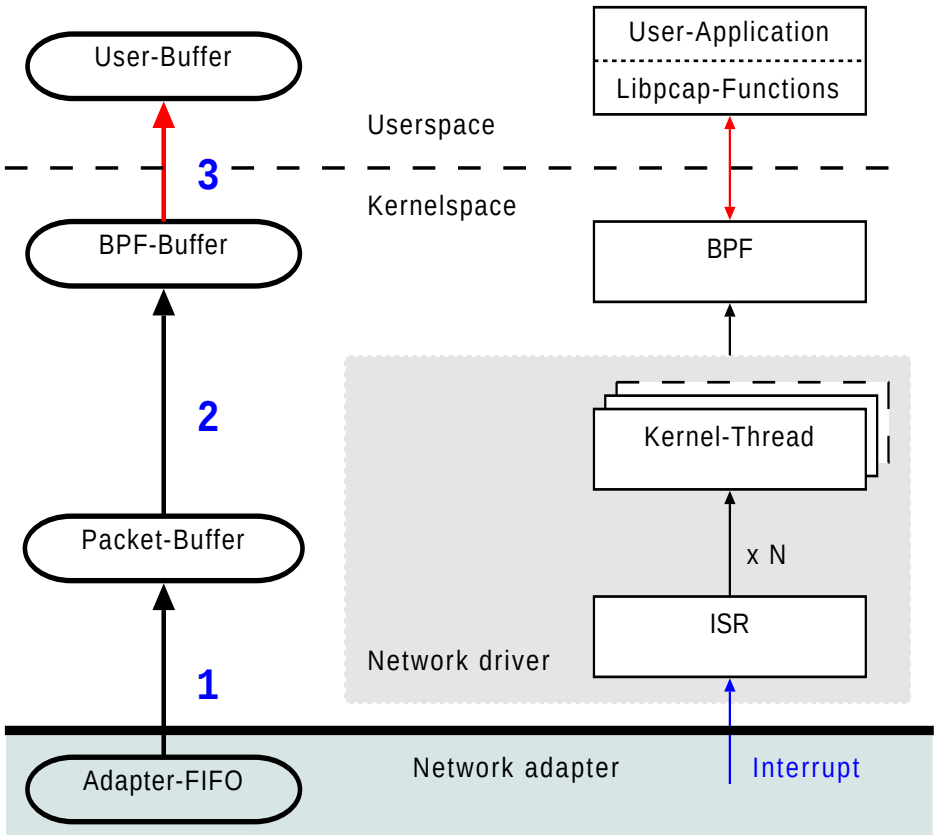
FreeBSD Standard Packet Capturing Stack

- Packet filtering
- Interrupt Service Routine
- Interrupt
- DMA-Transfer
- Packet is received and saved in Adapter-FIFO



FreeBSD Standard Packet Capturing Stack

- Access packets
- Packet filtering
- Interrupt Service Routine
- Interrupt
- DMA-Transfer
- Packet is received and saved in Adapter-FIFO



Disadvantages of Standard Packet Capturing Stack

- **Three copies per packet**
  1. DMA: *FIFO* $\Rightarrow$ *Packet Buffer*
  2. *Packet Buffer* $\Rightarrow$ *BPF Buffer*
  3. *BPF Buffer* $\Rightarrow$ *Userspace Buffer(*)*
- **Memory allocations**
  - For each new received packet will be a new *mbuf* allocated
- **System calls**
  - User-space application receives the packets using read(2)*(*)*
  - Saving packets to the hard disk

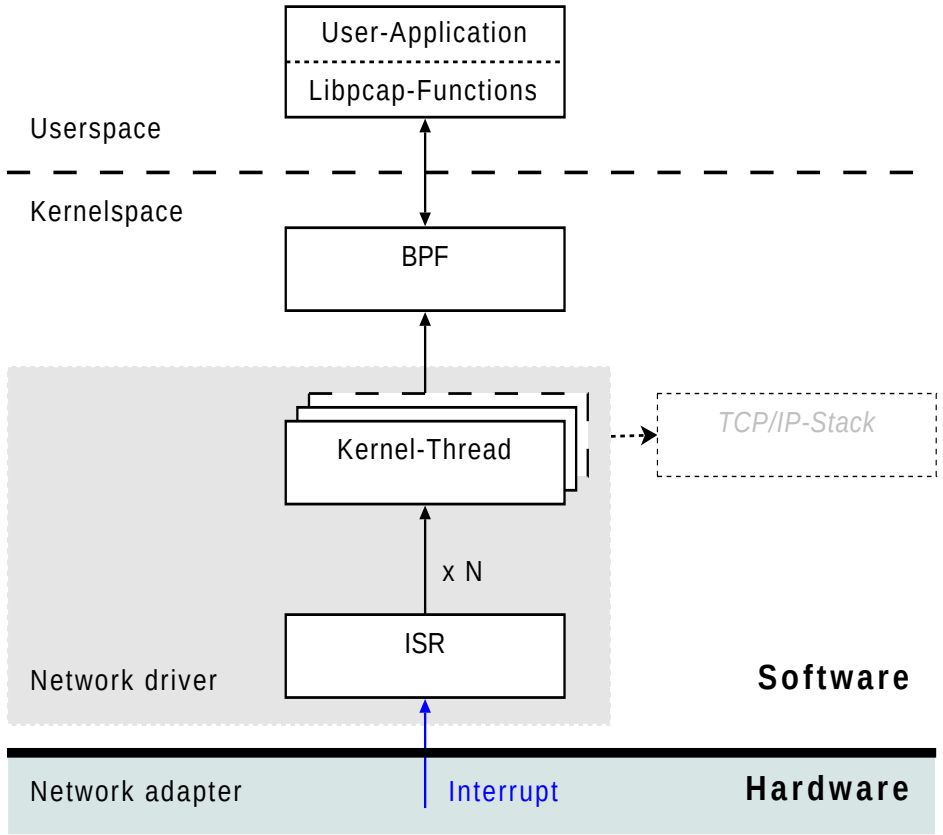
(*) Using Zero-Copy BPF eliminates the last copy and system call

Solution

Ringmap Packet Capturing Stack

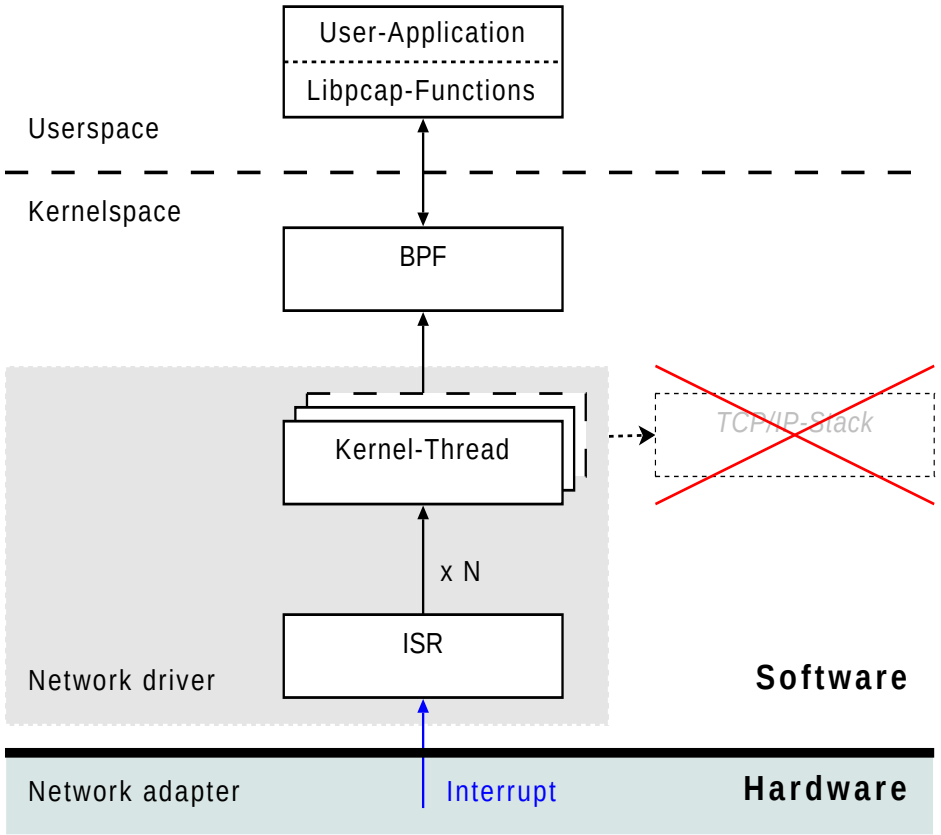
- **Ringmap**: The new packet capturing stack
- based on standard FreeBSD packet capturing stack:
  - based on generic network drivers and *libpcap*
    - but slightly changed.

What did we change in *generic*



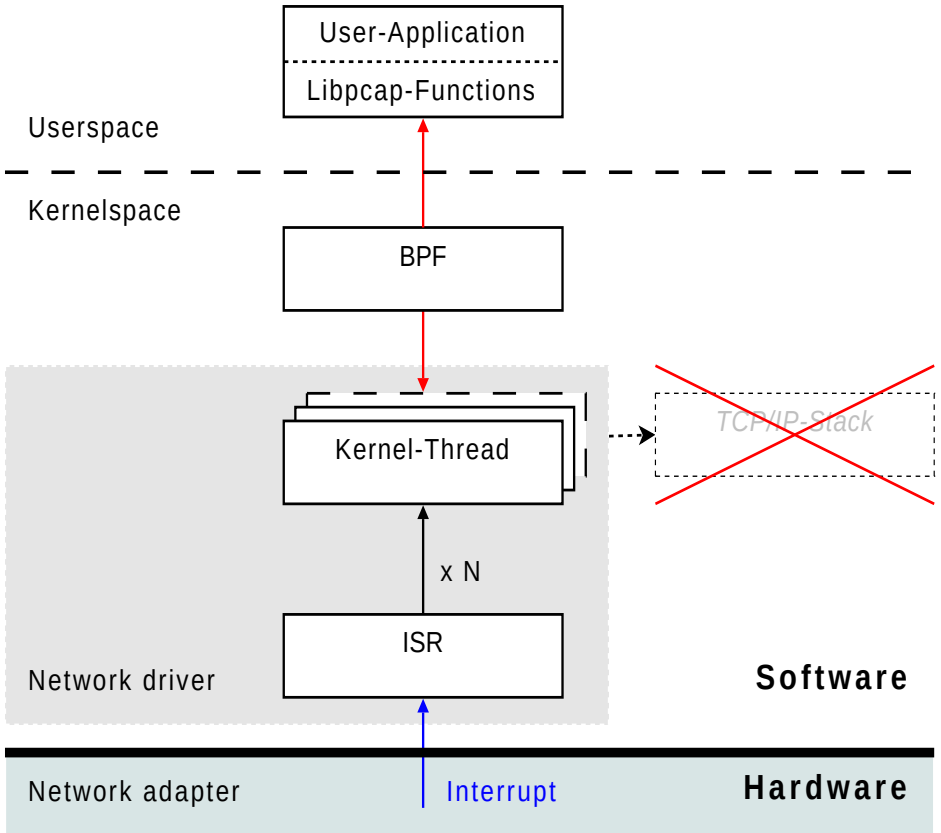
What did we change in *generic*

- Disabled TCP/IP-Stack



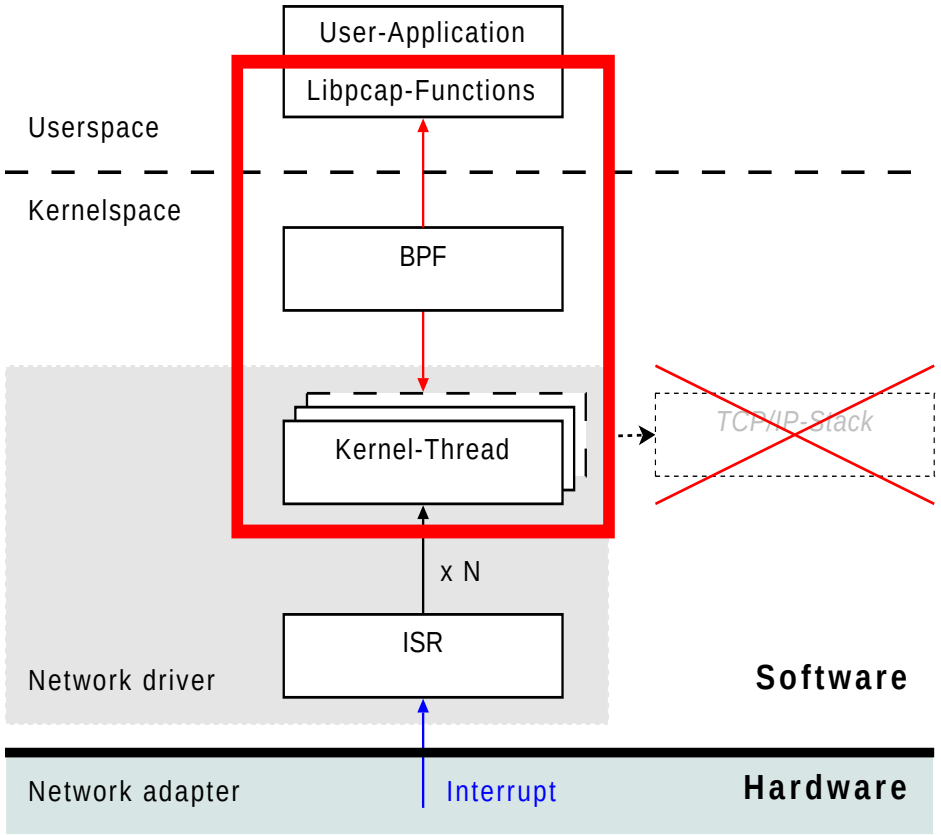
What did we change in *generic*

- Disabled TCP/IP-Stack
- BPF is accessible in both kernel and user-space



What did we change in *generic*

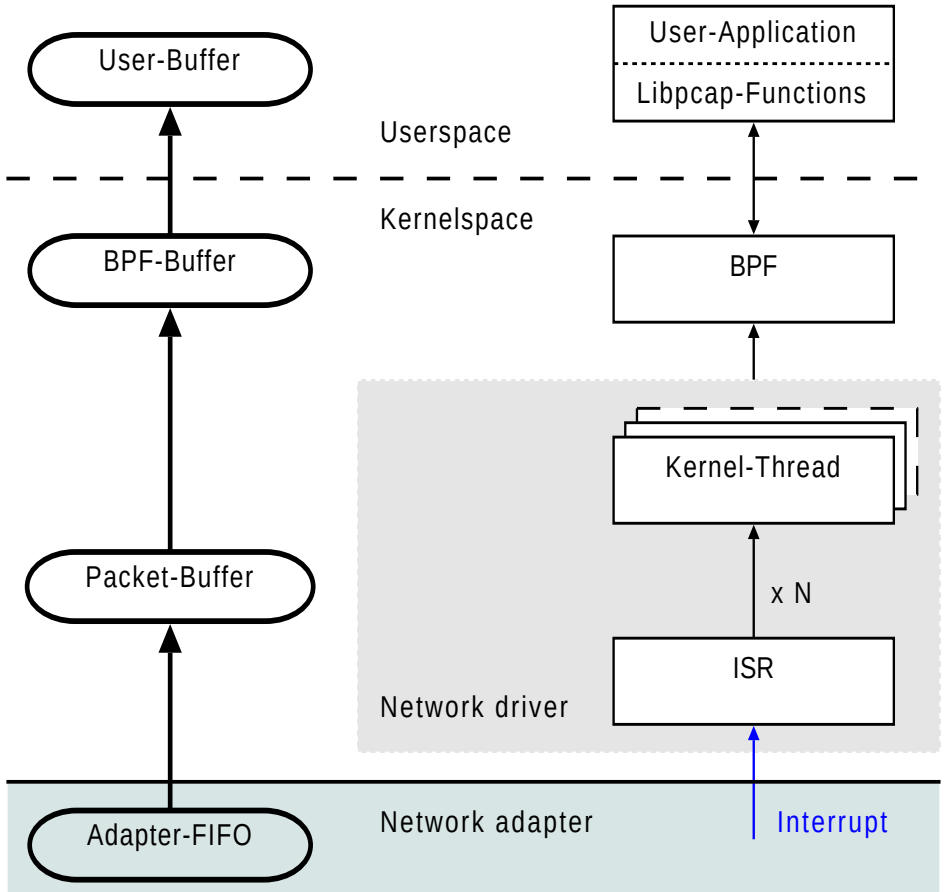
- Disabled TCP/IP-Stack
- BPF is accessible in both kernel and user-space
- Network driver and Libpcap is modified
  - BPF is unchanged



From *generic* to *ringmap*

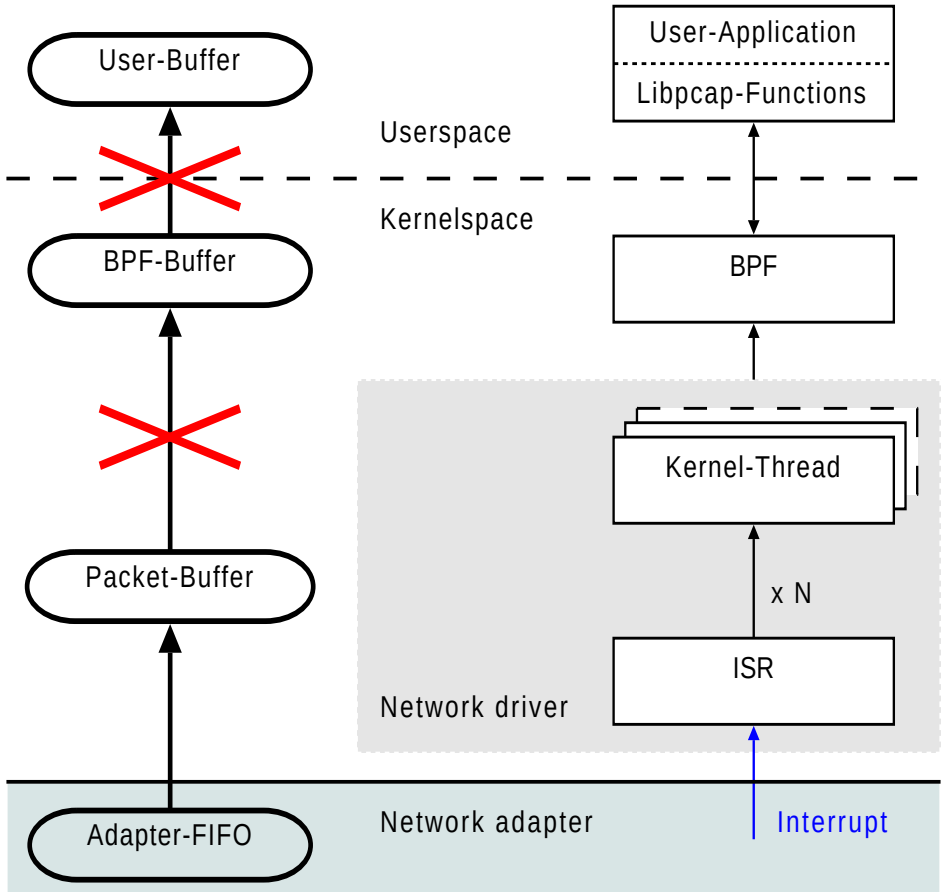
From generic to ringmap

- **generic**



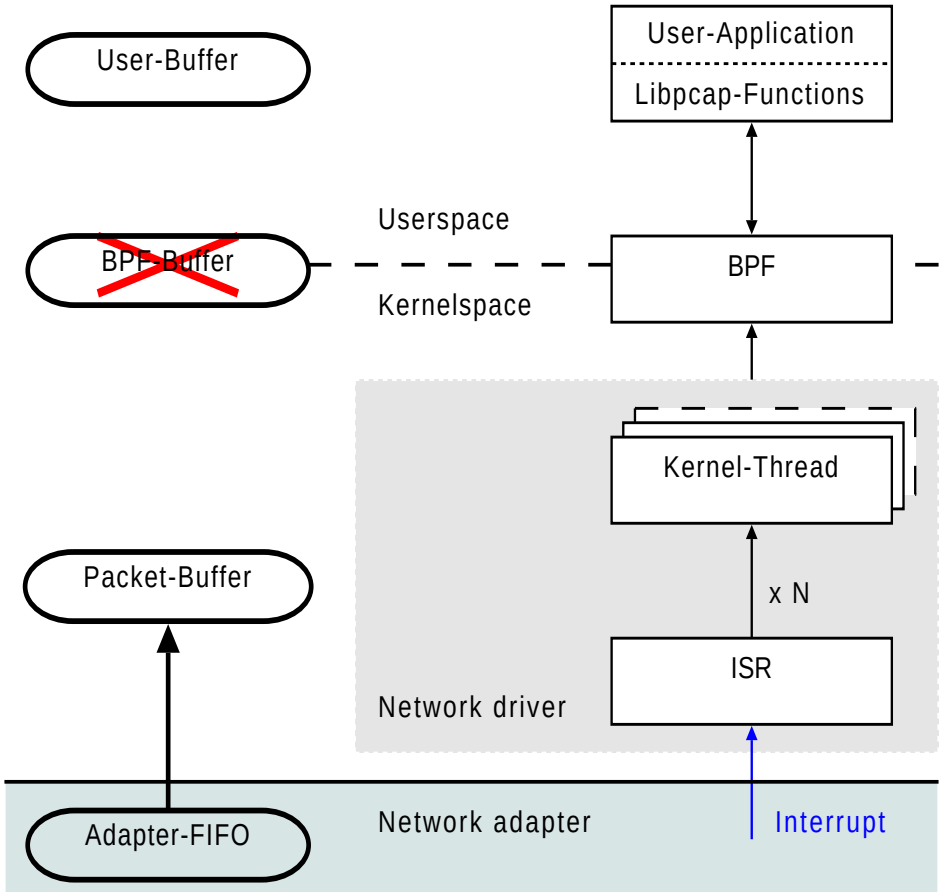
From generic to ringmap

- **generic**
- Eliminate packet copies and syscalls



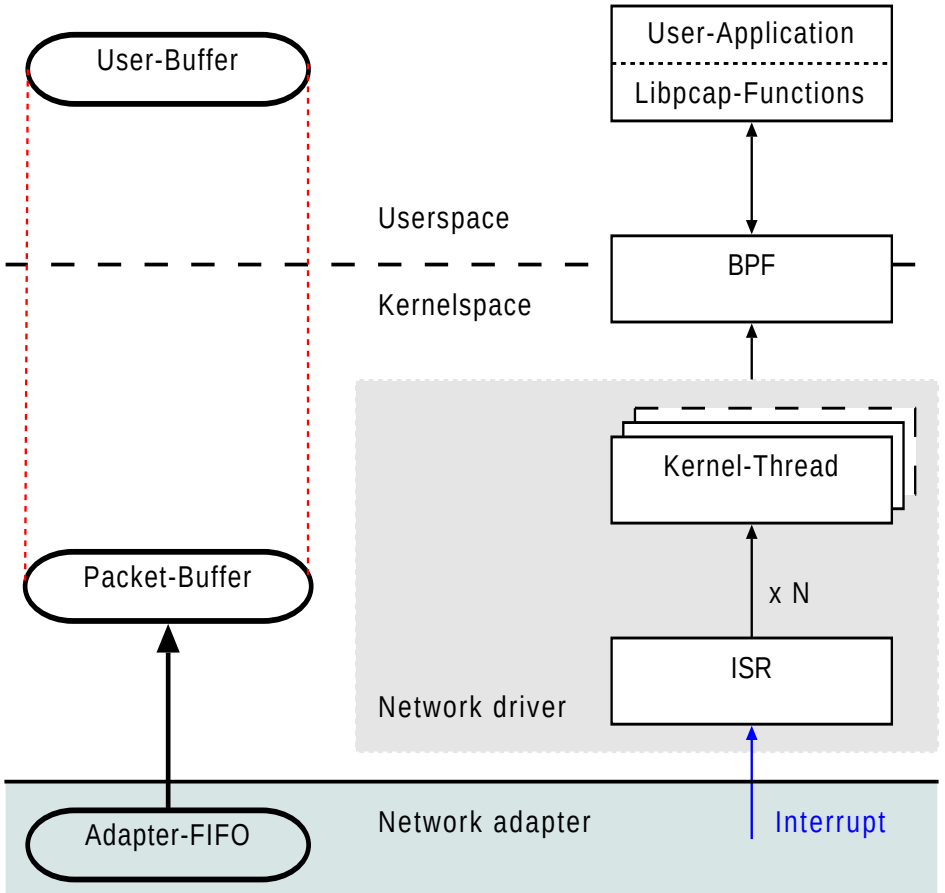
From generic to ringmap

- **generic**
- Eliminate packet copies and syscalls
- BPF-Buffer is then not necessary
  - Packet filtering is possible in both kernel and user-space



From generic to ringmap

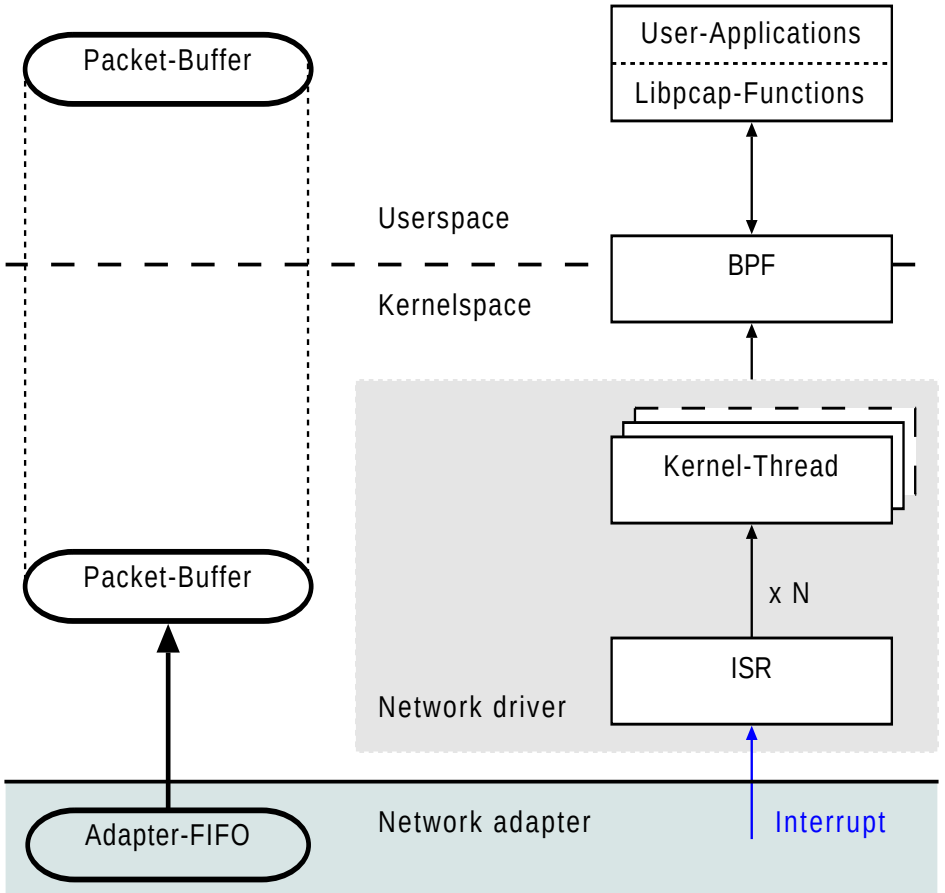
- **generic**
- Eliminate packet copies and syscalls
- BPF-Buffer is then not necessary
  - Packet filtering is possible in both kernel and user-space
- Mapping the Packet-Buffer to user-space



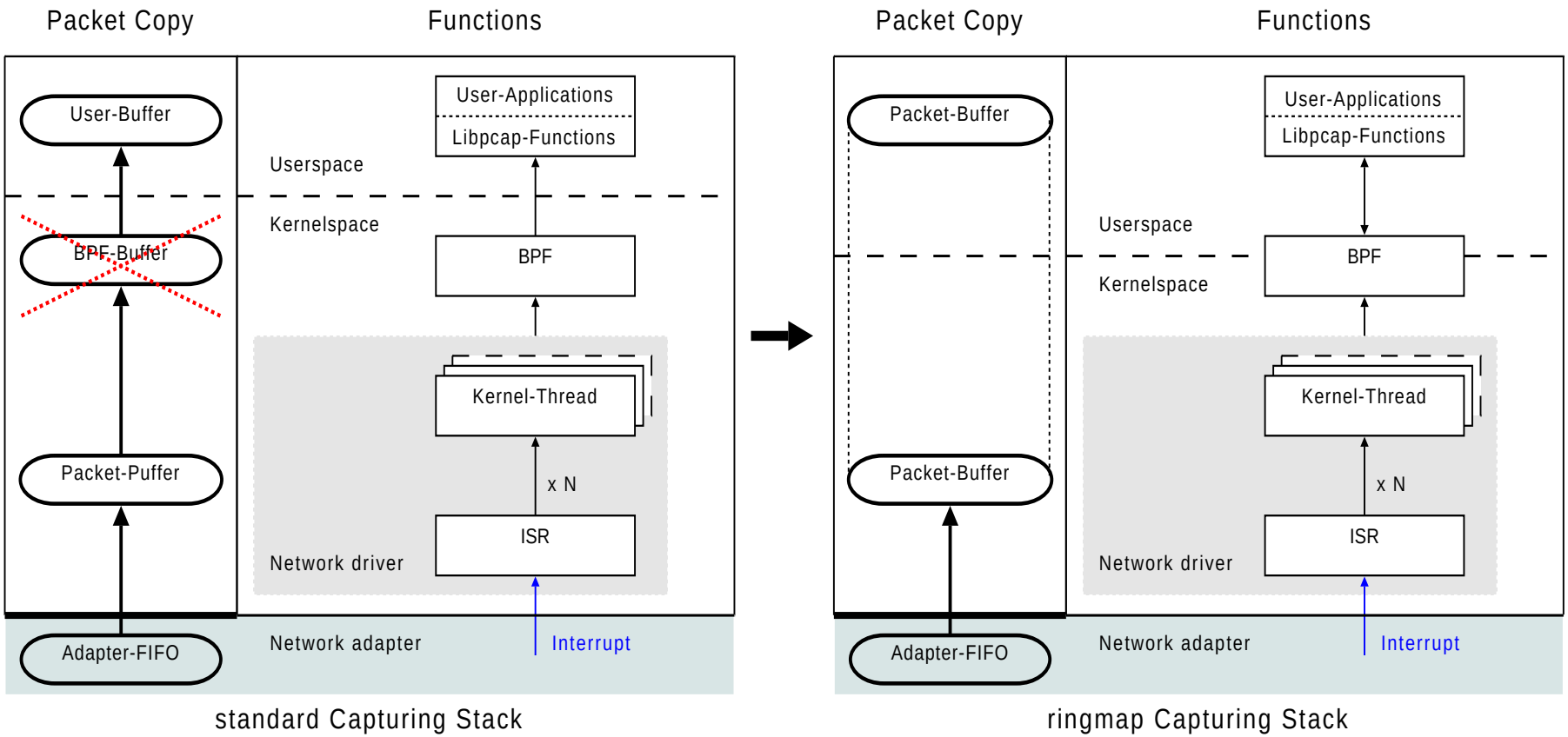
From generic to ringmap

- **generic**
- Eliminate packet copies and syscalls
- BPF-Buffer is then not necessary
  - Packet filtering is possible in both kernel and user-space
- Mapping the Packet-Buffer to user-space

$\Rightarrow$ **ringmap**



Overview



standard Capturing Stack

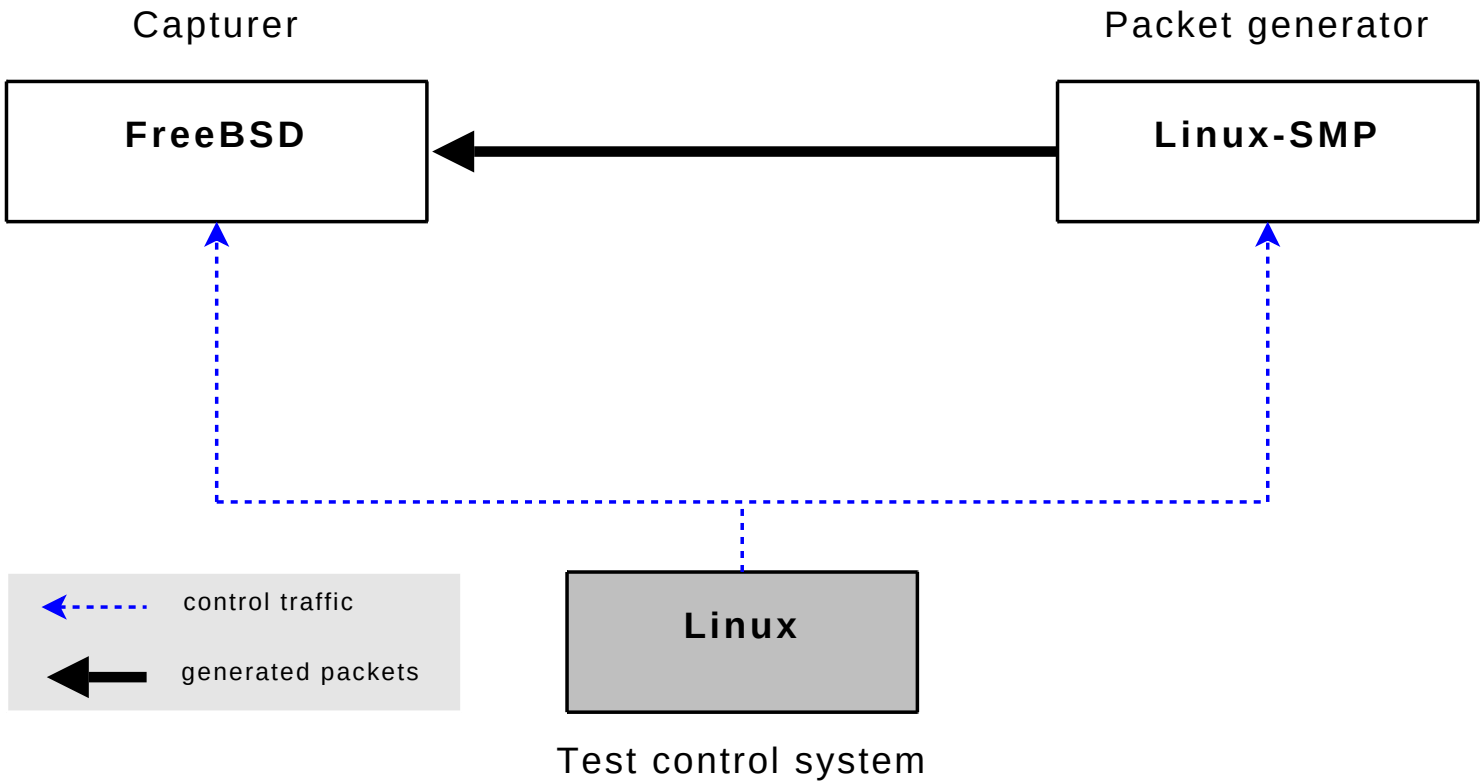
ringmap Capturing Stack

Measurements and performance evaluation

Goal of measurements

- Conducting performance evaluations of **ringmap** capturing stack
  - CPU-Load and packet loss during capturing
- Benchmarking: **generic** vs. **ringmap**

Testbed


Capturer
Packet generator
FreeBSD
Linux-SMP
control traffic
generated packets
Linux
Test control system

Measurement setup

Traffic generation

- Linux Kernel Packet Generator (*pktgen*)
  - generates network packets at very high speed with different:
    - packet sizes
    - bit-rate and packet-rate

Capturing

- **ringmap** and **generic** stacks
- Libpcap-application
  - accessing packets through call *pcap_loop()*
  - counting the captured packets

Calculation of results

- System load
  - Percentage proportion of time the CPU spent in system mode
  - Interrupt load is not considered
    - because of interrupt-throttling it is always constant for **generic** and **ringmap**
  - $syst = 1 - intr - user - nice - idle$
- Packet loss
  - Difference between generated and captured packets
  - $Packets_{loss} = Packets_{send} - Packets_{received}$

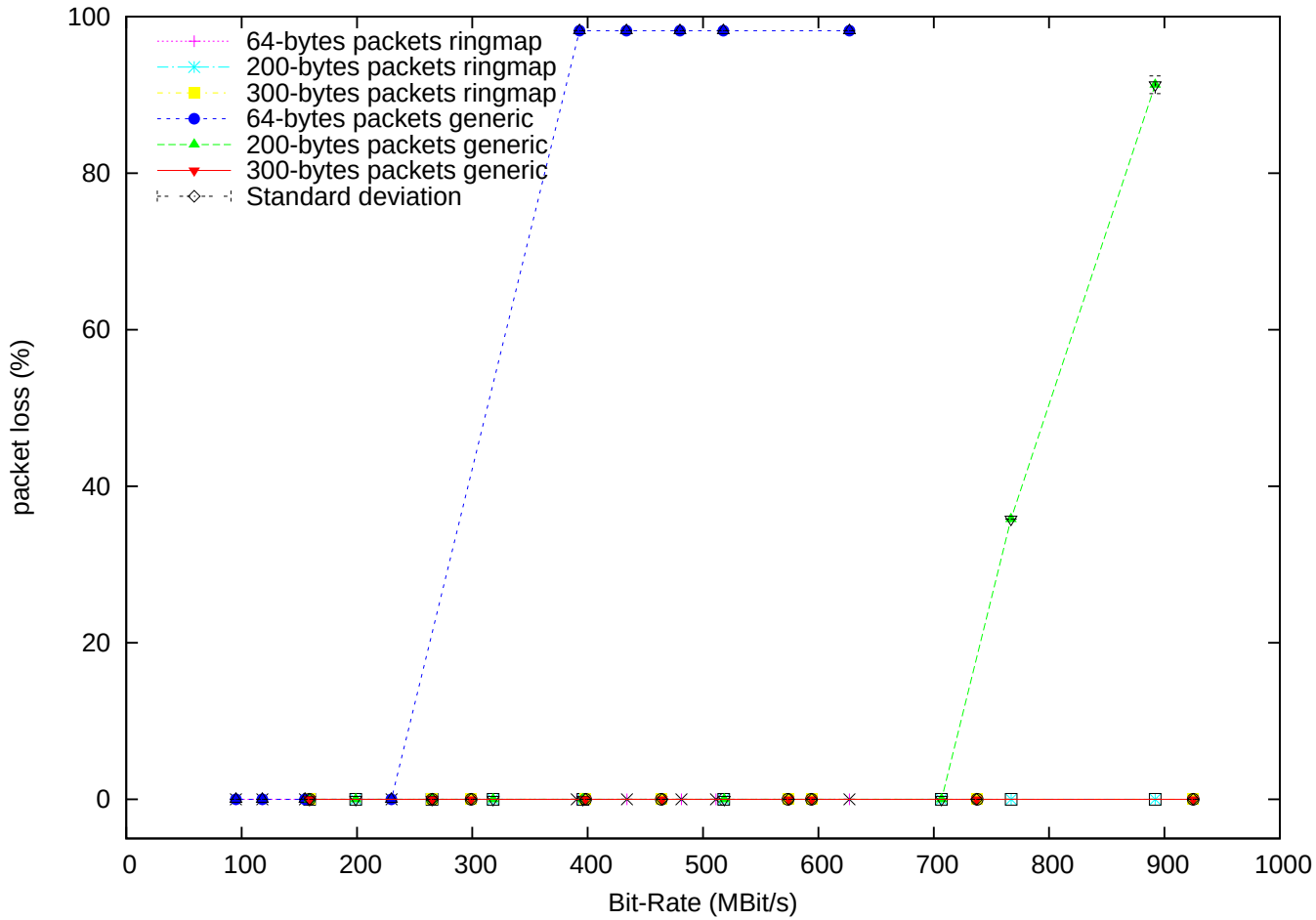
Testing sequence

1. Login to the capturer to
   - start of capturing
   - start of system load measurement applications
2. Login to the Packet-Generator to
   - generate traffic with:
     - a certain number of packets
     - a certain bit-rate and packet-rate
3. Login to the capturer to
   - stop capturing- und measurement-applications
4. Storing of measured values

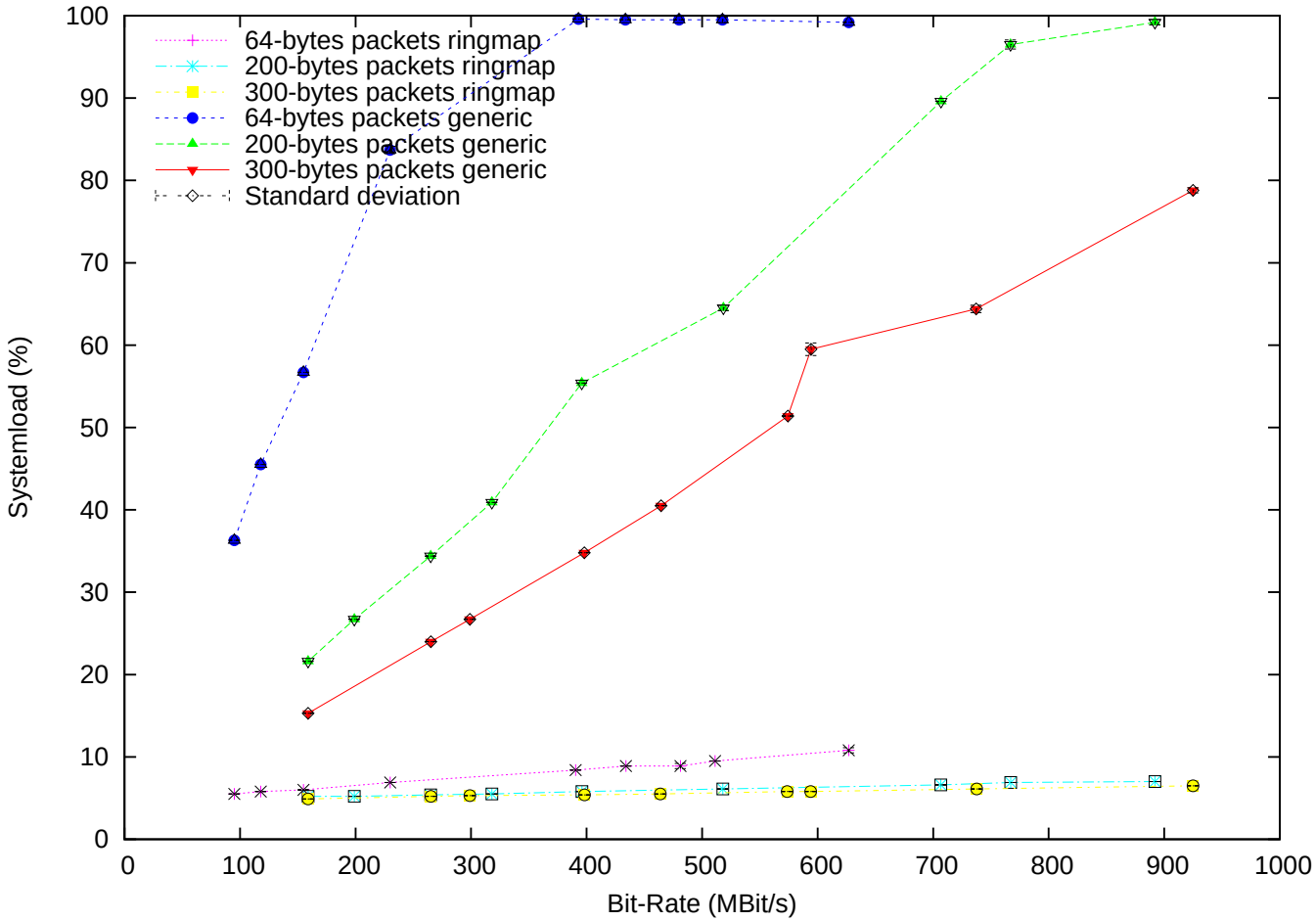
- Each experiment is repeated five times
- The mean and standard deviation is calculated

Results

generic vs. ringmap: Packet loss



generic vs. ringmap: System-load



Summary

Achieved goals

- **Enhanced Capturing-Performance**
  - very low system load (below 12%) and very low packet loss (below 0.02%)
    - only during capturing smallest 64-Bytes packets
    - for maximal reached packet-rate ($> 450000 pkts/sec$)
- **Stability and usability of implemented software**
  - during all experiments, no *kernel panics* and no *segmentation faults* occurred
  - very simple install and remove
    - two Shell-Scripts to installing and removing the *ringmap* Capturing Stacks
  - Libpcap-applications don't require modification in order to run with the *ringmap*

Achieved goals 2

- **Ringmap is easily portable to the other network controllers**
  - The software contains hardware dependent and hardware independent parts. Only hardware dependent part require modifications.
  - The generic driver and libpcap should contain a few hooks for calling *ringmap*-functions.
- **Packet Filtering**
  - Packet filtering can be accomplished using both *libpcap*- and kernel-BPF.

Achieved goals 3

- **Multithreaded Capturing**
  - Multiple applications can capture 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.

Future works

1. Benchmarking: **Zero-Copy BPF** vs. **ringmap**
2. Support for hardware time stamping
3. Using TCP/IP stack together with **ringmap**
4. 10-GbE-Paket-Capturing: Multiqueue + hardware support for packet filtering
5. Setting **ringmap** for packet transmission