Front. Electr. Electron. Eng. China 2010, 5(1): 36–42
DOI 10.1007/s11460-009-0070-6

RESEARCH ARTICLE

**Fan YANG, Yinan DOU, Zhenming LEI, Hua YU**

# Performance analysis of high rate packet capture on multiprocessor platform



**Abstract** With the continuous improvement of transmission rate, high speed network links require good performance of packet capture. The multiprocessor platform has strong computational capabilities, and brings new chance for high rate packet capture. In this paper, we analyze the performance of common packet capture approaches that are based on general-purpose multiprocessor platform. The analysis contains two aspects: one is the maximum packet capture rate and throughput on multiprocessor platform, the other is central processing unit (CPU) load under the maximum capture rate. By comparing and analyzing the experimental result, we give the maximum packet capture rate and throughput of different capture approaches. Furthermore, we analyze the CPU load which is produced by two capture processes run on the multiprocessor platform simultaneously and make a comparison with the single capture process.

**Keywords** multiprocessor platform, packet capture, high rate, performance

## 1 Introduction

Packet capture on high speed network links is the most fundamental and important technique in security auditing, intrusion detection, user behavior analysis, and many other fields. The capture performance directly influences the accuracy and effectiveness of network applications. Mass network data need high capture performance and rich computing resources.

The multiprocessor platform is more and more widely applied in the computer network field because of its strong computing capability and parallelism. Among numerous kinds of multiprocessor platforms, symmetrical multi-processing (SMP) is the most popular architecture. SMP aggregates multiple processors in one computer with the system bus, and shares memory subsystem. Although multiple processors are used, the operating system can balance the tasks and loads among them. This results in great improvement on data processing [1].

Many network servers use multiple processors as the performance assurance. Confronted with high speed network, packet capture performance has become the focus. Previous researches have showed some results of capture performance in specific scenarios [2,3], but the analysis of capture performance on multiprocessor platform is few. In this paper, we set up a multiprocessor environment using Linux operation system (OS), and analyze the performance of common packet capture approaches on this platform. This research achievement can not only help us to improve network packet capture capability on multiprocessor platform, but also contribute to finding the bottlenecks of some network applications.

The rest of this paper is organized as follows. Section 2 states the principles of common packet capture approaches. Section 3 describes our experimental environment and methodology. Section 4 analyzes the experimental results. Section 5 presents the final conclusion and future work.

## 2 Common packet capture approaches on high speed links

In current traffic monitoring applications, socket capture, libpcap capture and zero-copy capture are commonly adopted. In this section, we will briefly describe the principles of these types of capture approaches so as to analyze performance better.

### 2.1 Socket capture

Socket can be considered as a very important concept in the network field [4]. PF_INET and PF_PACKET are two protocol suites that are usually used in socket capture. The packet captured by PF_PACKET protocol suite contains

Received June 25, 2009; accepted September 10, 2009

Fan YANG (✉), Yinan DOU, Zhenming LEI, Hua YU
School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China
E-mail: yangfangood@gmail.com

the complete data from the media access control (MAC) header. On the contrary, for PF_INET protocol suite, the receiving application can only get the packet data begins from the Internet protocol (IP) header rather than the whole packet. In order to obtain the whole packet information from network links, we only analyze the socket capture performance using PF_PACKET protocol suites.

PF_PACKET protocol suite is not added into Linux until kernel version 2.2. This protocol suite permits user applications to receive packets by network interface card (NIC) driver programs, and it can avoid the processing done by protocol stack. When we select PF_PACKET protocol suite to capture packets, we can receive the entire Ethernet frame in user application buffers.

### 2.2 Libpcap capture

As a system-independent application programming interface (API) for user-level packet capture under Linux environment, libpcap is the foundation of many network monitoring applications [5]. It provides a portable framework for traffic monitoring. Libpcap is mainly composed of two parts: network tap and packet filter. Network tap collects data from the NIC driver, while packet filter determines whether to receive the packet. Libpcap uses BSD packet filter (BPF) to filter the packets. BPF can decide whether to accept the packet through matching user-defined rules. If the packet can be accepted, it will be passed into user application buffers. If no rules are set, all the packets will be transferred to user application buffers.

### 2.3 Zero-copy capture

By analyzing PF_PACKET socket and libpcap capture approaches, we find that traditional packet capture techniques must copy the packet data twice from the NIC to user space application buffers. One occurs from NIC to kernel space and the other occurs from kernel space to user space [6]. Besides two copies, it needs system calls to complete packet reception. As we know, memory copies and system calls cost much time. Therefore, frequent memory copies and system calls will be the performance bottlenecks when packet arriving rate is very high. Zero-copy capture technique can reduce data copies and system calls which are in the packet process from NIC to user space [7,8]. It also can alleviate the central processing unit (CPU) load and make CPU free from memory copies and system calls. All these advantages mentioned above will result in a performance improvement of network packet capture. Figure 1 shows the comparison of traditional capture and zero-copy capture.

Zero-copy capture procedures can be described as follows:

1) NIC receives the packet, and directly sends it to a buffer in the kernel by direct memory access (DMA).

2) Map the kernel buffer into user space.

3) User space application takes out the packet from the mapped buffer.

## 3 Experimental environment and methodology

Based on the analysis above, we carry out an experiment on a DELL PowerEdge 2950 multiprocessor server. The specification of this server is as follows:

1) Dual Intel Xeon 2.80 GHz processors;

2) 1 GB memory;

3) E1000 Gigabit NIC.

We choose Cent OS 4.4 (kernel 2.6.9) as our experiment OS. In this experiment, we use Smartbits 6000 for generating high rate packets. In order to ensure the accuracy, we adopt the method of taking an average result of several tests.

On the multiprocessor platform, we accomplish the experiment on the following aspects: measuring the performance of different packet capture approaches; analyzing the influence of packet capture on CPU load.

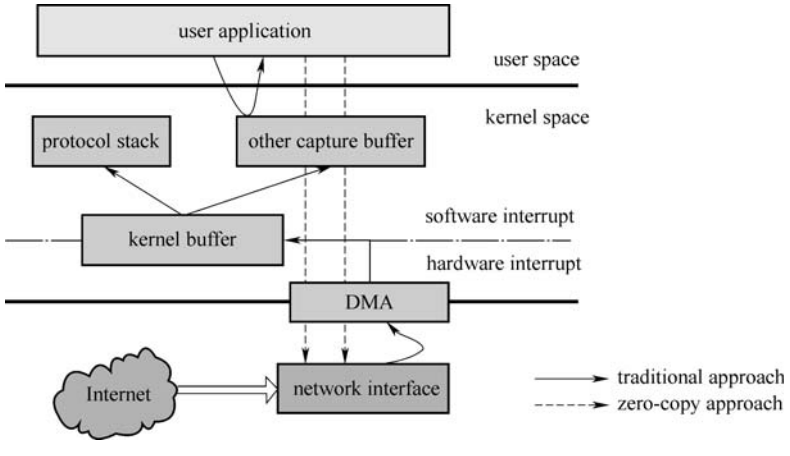


**Fig. 1** Comparison of traditional capture approach and zero-copy capture approach

In order to find the performance differences on multiprocessor platform, in each experiment part, we execute single capture process and dual capture processes separately.

For the sake of brevity, in the following sections, PF_PACKET, libpcap, and zero-copy are separately short for the PF_PACKET socket capture approach, libpcap capture approach, and zero-copy capture approach.

## 4 Experimental results analysis

### 4.1 Maximum capture rate and throughput of various capture approaches on multiprocessor platform

#### 4.1.1 Single capture process

For the capture approaches mentioned above, we measure the maximum capture rate and throughput without packet loss. The results are shown in Tables 1 and 2.

Tables 1 and 2 illustrate:

1) With packet length increases, the receiving rate (packet/s, pps) decreases but total throughput (bit/s) rises.

2) In multiprocessor platform, the capture capability using socket and libpcap already meets the basic requirements of common network application. This conclusion is different from some previous researches which deemed that socket and libpcap are of low receiving performance.

3) Zero-copy capture performance is much better than the other approaches. However, zero-copy technique bypasses the kernel protocol stack, and results in the packet which cannot be processed by protocol stack.

#### 4.1.2 Comparison between dual capture processes and single capture process

For some network applications, they must receive packets from more than one link. In order to meet this requirement, we execute two capture processes on the multiprocessor server simultaneously and measure the capture performance without packet loss. We make a comparison between two capture processes and single capture process so as to give a better analysis. Taking PF_PACKET and libpcap for example, Figs. 2 and 3 show the comparison effect of maximum packet rate and throughput, respectively.

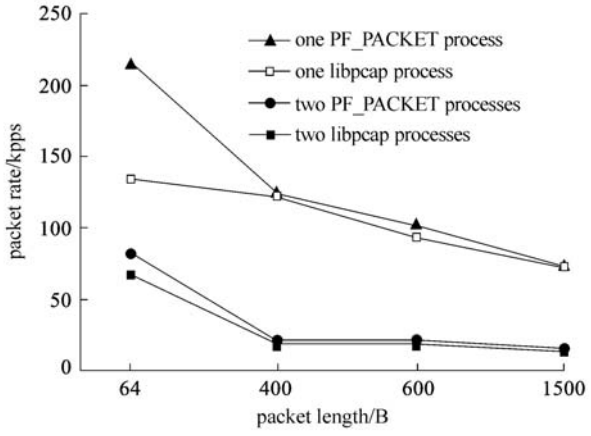


**Fig. 2** Comparison of maximum packet rate between one capture process and two capture processes

From Figs. 2 and 3, we can infer:

1) In the case of no packet loss, as packet length increases, capture rate drops off and the throughput grows monotonically.

2) When two capture processes are executed simultaneously, packet will be dispatched in Linux kernel protocol stack, and will be copied twice from kernel space to user space. This is why the performance of dual processes slumps apparently.

3) In Fig. 2, for 64 B packet, single capture process can reach the rate of 130 kpps, while dual capture processes

**Table 1** Maximum packet rate with one capture process separately

| length/B | PF_PACKET/(packet·s$^{-1}$) | libpcap/(packet·s$^{-1}$) | zero-copy/(packet·s$^{-1}$) |
|---|---|---|---|
| 64 | 215516 | 132979 | 947252 |
| 400 | 123032 | 121596 | 286754 |
| 600 | 102460 | 92455 | 195658 |
| 1500 | 72805 | 70384 | 81158 |

**Table 2** Maximum throughput with one capture process separately

| length/B | PF_PACKET/(bit·s$^{-1}$) | libpcap/(bit·s$^{-1}$) | zero-copy/(bit·s$^{-1}$) |
|---|---|---|---|
| 64 | 131034483 | 80851064 | 575745257 |
| 400 | 422131147 | 400778210 | 927981693 |
| 600 | 501639344 | 452662722 | 950388741 |
| 1500 | 870967742 | 851351351 | 979211387 |

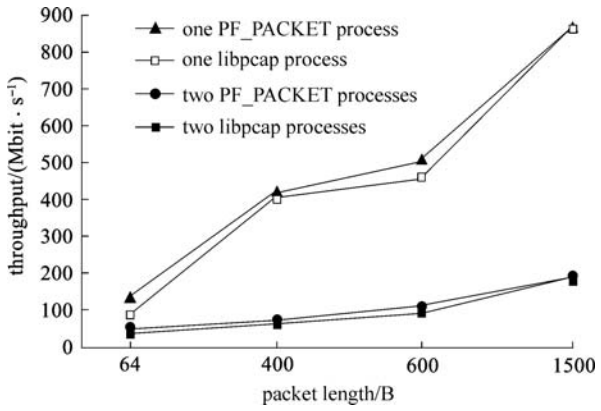


**Fig. 3** Comparison of maximum throughput between one capture process and two capture processes

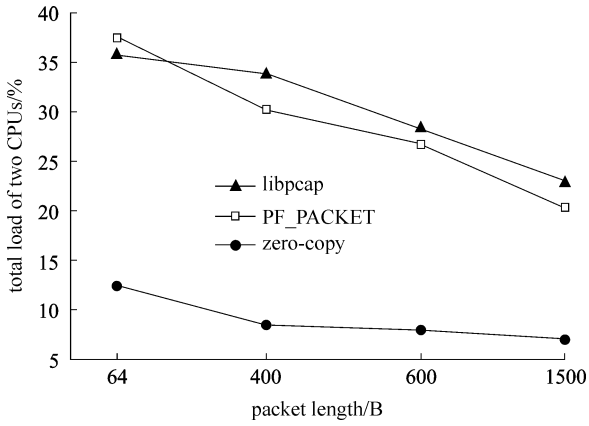


**Fig. 4** Two CPUs' total load of single capture process at the maximum capture rate

rate is only approximately 70 kpps, which is about half of the single process.

4) From Fig. 3, by considering the throughput, we find that the performance advantages of single capture process are not visible for 64 B packet. But for the length of 1500 B, the throughput of single capture process is about four times of dual processes.

## 4.2 CPU load of various capture approaches under the maximum capture rate

### 4.2.1 CPU load of single capture process

We measure the total load of two CPUs with executing single libpcap, PF_PACKET, and zero-copy capture processes individually. All the three processes are running at the maximum capture rate without packet loss. Figure 4 shows the result.

Figure 4 tells us that a short length packet puts a heavier pressure on CPUs for the three different capture approaches.

As the packet length grows, the total load produced by PF_PACKET and libpcap decreases much sharper than zero-copy. Since zero-copy gives the CPUs a low pressure, it can maintain a steady low level of total load.

We can find that there are no apparent differences between libpcap and PF_PACKET approaches on two CPUs' total load, and this feature consists with the capture performance analysis in Sect. 4.1.1.

In the remainder of this section, we make a more in-depth load analysis of each CPU on multiprocessor platform. Figures 5–7 show the load for CPU0 and CPU1, which separately runs single libpcap, PF_PACKET, and zero-copy capture processes. We also divide CPU load into kernel consumed part and user consumed part.

From the statistics in Figs. 5–7, we know that Linux attaches the single capture process to one CPU during the whole receiving time. The advantage of this scheduling method is that there is almost no process switching cost

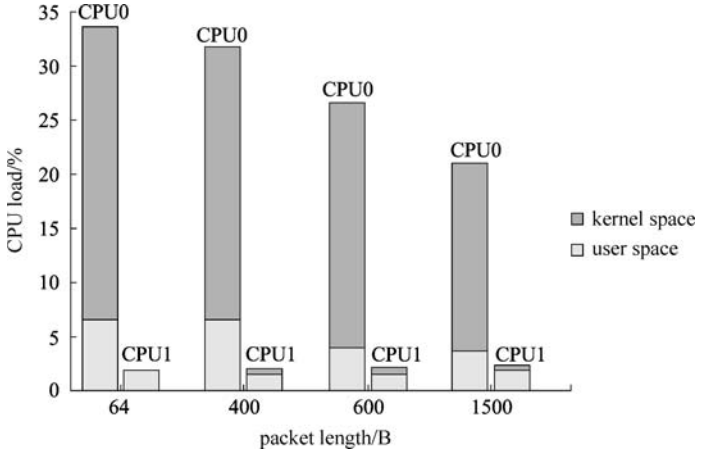


**Fig. 5** CPU0 and CPU1's load of single libpcap process at the maximum capture rate

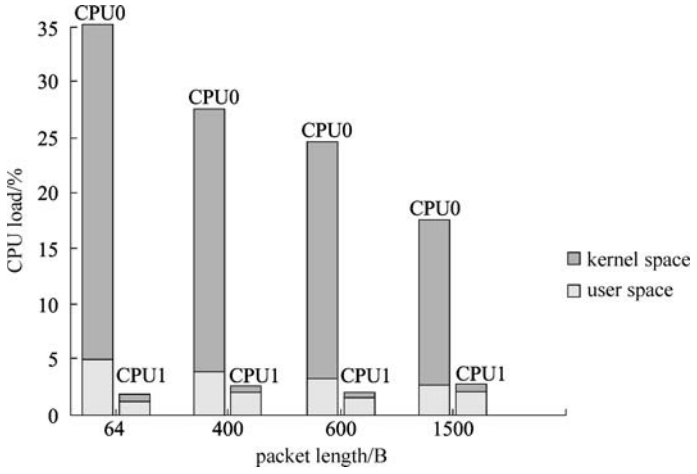


**Fig. 6** CPU0 and CPU1's load of single PF_PACKET process at the maximum capture rate

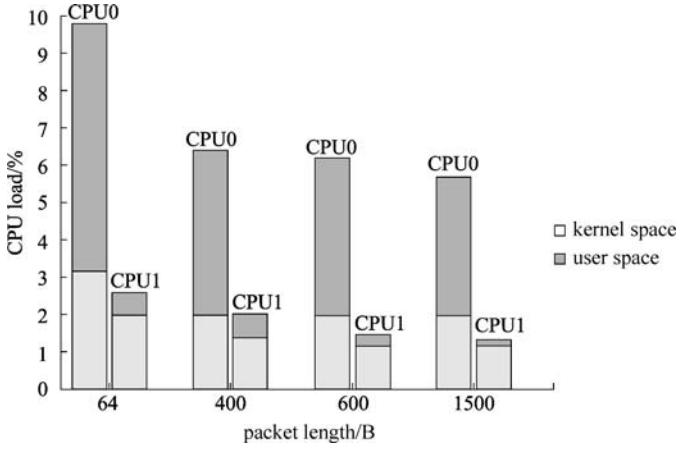


**Fig. 7** CPU0 and CPU1's load of single zero-copy process at the maximum capture rate

when running single process for packet receiving. However, Figs. 5–7 also show that packet capture process does not sufficiently utilize the CPUs resources so that one processor is almost idle in most of the capture time.

When capture process is attached to one CPU, the CPU cache missing rate is lower; therefore, it has a higher efficiency.

Processor resources are mainly consumed in kernel space. Through the description of packet capture principle in Sect. 2, we know these packet capture processes are mainly executed in kernel space, and the experimental results here confirm it.

### 4.2.2 CPU load of dual capture processes

In order to explore the multiprocessor platform capacity, we run two capture processes on the multiprocessor server and test the influence on two CPUs' total load. We take PF_PACKET and libpcap as example and both of them are running at the maximum capture rate. The results are shown in Fig. 8.

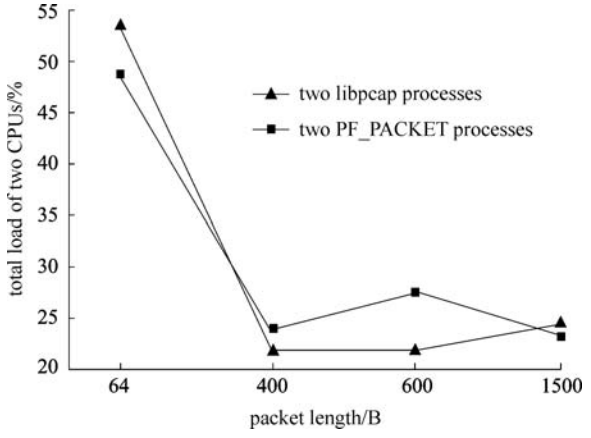


**Fig. 8** Comparison of two CPUs' total load between two libpcap processes and two PF_PACKET processes

As Fig. 8 shows, two CPUs' total load does not take on the monotonic trend. That is because when using two processes to capture packets, the operating system will roughly balance the whole workload on two CPUs. At the same time, continuous process switching makes the cache content refresh rapidly and results in that the cache cannot keep related data for the same CPU. Therefore, cache missing rate grows sharply and badly affect the capture performance. This negative effect results in the non-monotonic trend of two CPUs' total load in Fig. 8.

When packet length increases from 64 B to 400 B, the total load decreases rapidly, and this is significantly different from the single capture process.

We make a more in-depth load analysis of each CPU on multiprocessor platform. Figures 9 and 10 show the load for CPU0 and CPU1, which separately runs dual libpcap and PF_PACKET capture processes. We also divide the CPU load into kernel consumed part and user consumed part.

By considering Figs. 9 and 10, we can infer that when running double processes to receive packets, two CPUs will balance the workload approximately. However, the balance is not absolutely fair. For example in Fig. 9, the system does not balance the computing resources well for the case of 64 B and 400 B packet.

When the operating system balances workload and schedules processes, a large number of switching costs are brought to processors, which makes kernel space consumed load become larger. This feature increases the overhead of the whole multiprocessor system.

## 5 Conclusion and future work

The fast development of multiprocessor platforms opens up a broader space for the innovation of network traffic monitoring. In this paper, we deeply analyzed the performance of several common packet capture approaches on the general-purpose multiprocessor platform and have drawn some new conclusions. We emphasized on the

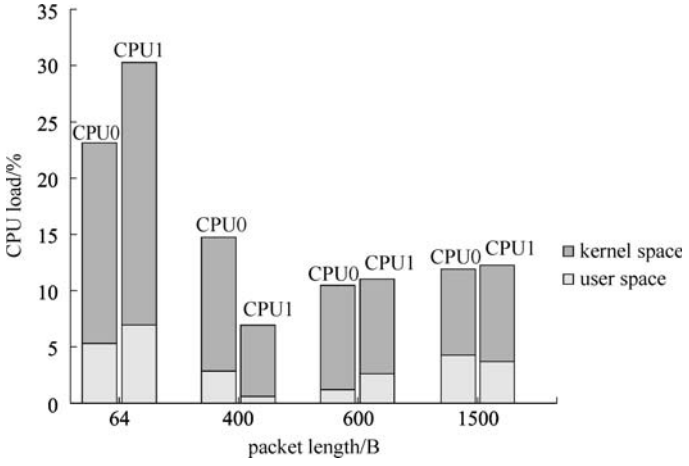


**Fig. 9** CPU0 and CPU1's load of dual libpcap processes at the maximum capture rate

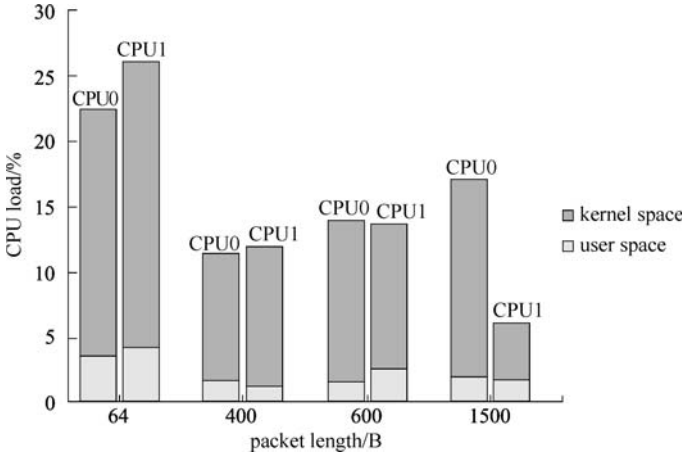


**Fig. 10** CPU0 and CPU1's load of dual PF_PACKET processes at the maximum capture rate

maximum capture rate and throughput of each capture techniques. We also did a detailed analysis of the workload pressure on CPUs in different conditions. All these were proved by abundant experimental results.

In the future, we will continue the research on the following aspects: 1) analyzing the influence of different packet filters on capture performance; 2) with the development of CMP platform, we plan to extend this research to multi-core platform, improving packet capture performance by making use of the new advantages.

**Acknowledgements** This work was supported by the Key Project in the National Science and Technology Pillar Program (No. 2008BAH37B04) and the Consolidated Project of IBM Institute of China and Beijing University of Posts and Telecommunications (No. JLP200906011-1).

## References

1. Mallik A, Zhang Y, Memik G. Automated task distribution in multicore network processors using statistical analysis. In: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for Networking and Communications Systems. Orlando: ACM Press, 2007, 67–76
2. Weigle E, Feng W C. Ticketing high-speed traffic with commodity hardware and software. In: Proceedings of Passive and Active Measurement Workshop 2002. Fort Collins: Agilent, 2002, 156–166
3. Donnelly S. DAG packet capture performance. Endace Limited. White Paper, 2006
4. Toll W E. Socket programming in the data communications laboratory. ACM SIGCSE Bulletin, 1995, 27(1): 39–43
5. Papadogiannakis A, Antoniades D, Polychronakis M, Markatos E P. Improving the performance of passive network monitoring applications using locality buffering. In: Proceedings of the 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. Istanbul: IEEE Computer Society Press, 2007, 151–157
6. Balaji P, Shivam P, Wyckoff P, Panda D. High performance user level sockets over Gigabit Ethernet. In: Proceedings of the International Conference on Cluster Computing. Chicago: IEEE Computer Society Press, 2002, 179–186
7. Liu T H, Zhu H F, Chang G, Zhou C S. Research and implementation of zero-copy technology in Linux. In: Proceedings of 2006 IEEE Sarnoff Symposium. Princeton: IEEE, 2006, 1–4
8. Halvorsen P, Jorde E, Skevik K-A, Goebel V, Plagemann T. Performance tradeoffs for static allocation of zero-copy buffers. In: Proceedings of the 28th Euromicro Conference. Dortmund: IEEE Computer Society Press, 2002, 138–143