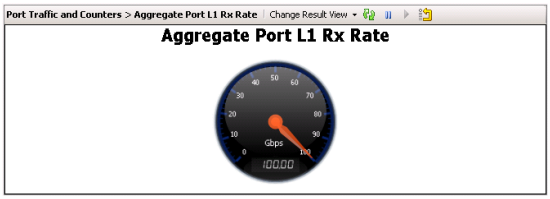
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

DPDK for COMBO Network Cards

Matej Vido*



Abstract
Software framework Data Plane Development Kit provides a standard API for fast packet processing in the user space. The DPDK covers multiple devices and architectures from different vendors. The CESNET association develops the family of COMBO network cards that are able to process traffic up to 100 Gb/s through their proprietary SZE2 interface. This paper describes how the SZE2 library can be used as a backend for the DPDK. This connection enables receiving and transmitting data through the COMBO network cards in a more standard way. The SZE2 library is utilized to create a user space Poll Mode Driver for the DPDK and it has already become part of the DPDK mainline in the version 2.2.0 (December 2015). The correct positioning of packets in DPDK message buffers and SZE2 buffers is accomplished by copying data between buffer memories. The performance benchmarks has shown that the COMBO-100G network card can receive and transmit over 140 millions of 64 B long Ethernet frames per second unidirectionally and a single port can handle up to 134 millions of 64 B long frames per second in the bidirectional traffic using multiple CPU cores.

Keywords: DPDK — COMBO Network Card — COMBO-100G — SZE2 — libsze2 — 100Gb/s Ethernet

Supplementary Material: *N/A*

*xvidom00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

The integral part of any traffic monitoring and analysis in current networking systems is the wirespeed packet capture. The IEEE 802.3ba-2010 standard for 40 Gb/s and 100 Gb/s Ethernet brings new challenges for the packet processing. At the link layer of the Ethernet protocol, the shortest packet is 64 B long. Transmission of such frame requires 84 B at the physical layer (additional 7 B for preamble, 1 B for start of frame delimiter, and 12 B for interpacket gap). The 100 Gb/s Ethernet can transfer 148,8 millions of packets per second (pps). To capture and analyse data at the 100 Gb/s wirespeed, the processor with a single core has to process one packet in 6,72 ns (i. e. 20 cycles on a CPU with the frequency of 3 GHz).

User space applications in the Linux-based operating system can process network data using sockets API and the packet capture library. Data transferred by these facilities are passed through the kernel networking stack which is too slow for the current needs and can hardly accommodate speeds up to 10 Gb/s. Therefore, frameworks which bypass the kernel and move the processing from the kernel space to the user space were developed such as Data Plane Development Kit (DPDK) [1], netmap [2], PF_RING [3] and OpenOnload [4]. These solutions support multiple operating systems and various hardware. Comparison of the frameworks is out of scope this paper and can be found in papers by Gallenmüller and others [5, 6]. Besides that, the CESNET association develops the

network cards COMBO which can transfer data to the user space through a proprietary interface SZE2 which is introduced and discussed in detail in the master's thesis by Jiří Slabý [7] and Andrej Hank [8]. This paper describes interconnection of the SZE2 interface with the DPDK in order to build applications for the COMBO network cards using a more widespread API.

The DPDK consists of multiple drivers and libraries for packet processing which can be done purely in the user space. The drivers might require a kernel module to control communication with network card through the PCI-Express bus and access the card registers. Packet data is transferred from a network card through the direct memory access (DMA) transfers and stored in a DPDK-specific message buffer (mbuf). The mbuf allows an easy manipulation with packets without a need of the additional memory copies when the packet is being processed by an application. More background on DPDK is described in section 2. The software side of the SZE2 interface consists of kernel modules and the user space libsze2 library. Data are transferred through the DMA transfers to a buffer structure consisting of several larger memory areas. This structure is referred as *circular buffer* in the further text and described in section 3. The packet data from multiple packets are aggregated into a continuous memory space. The transfer of consecutive packets requires less PCI-Express transactions than in case of the way used by DPDK. On the other hand manipulation with packets coming from the SZE2 interface is more limited. The SZE2 interface principles are mentioned in section 3.

This paper is focused on the implementation of a DPDK driver *szedata2* for COMBO network cards. The driver hides the libsze2 behind the DPDK API. On the receive side, data is copied from the SZE2 *circular buffer* to the mbuf memory, vice-versa for the transmit side. Section 4 describes the *szedata2* DPDK driver in detail. Section 5 provides the performance results obtained from benchmarks.

The DPDK driver *szedata2* enables creating applications on top of the COMBO network cards using the DPDK API. This setup scales up to 8 CPU cores and allows to utilize up to 100 Gb/s of the Ethernet bandwidth in unidirectional reception, 94 Gb/s in unidirectional transmission and 90 Gb/s in bidirectional traffic with the shortest packets.

2. Principles of DPDK

This section provides basic information about the DPDK and its principles that are essential for reaching the high performance. Detailed information can be obtained from the DPDK Programmer's Guide [9].

The DPDK supports multiple CPU architectures: Intel x86, IBM Power 8, EZchip TILE-Gx, ARM [1]. It contains drivers for various network cards from different vendors.

The Environment Abstraction Layer (EAL) is a DPDK core library which encapsulates architecture and system specifics. The EAL manages access to hardware resources and memory. The DPDK uses huge pages Translation Lookaside Buffers (TLB) support. Huge pages reduce TLB miss rates. The EAL controls allocation of memory from huge pages and allocation of continuous physical memory blocks (called *memzones*) that are used for DMA transfers.

Packets in DPDK are stored in mbufs. The mbufs are allocated from DPDK memory pools (called *mempool*) which manage lockless queues of fixed-sized objects and ensure suitable distribution of mbufs through memory channels. The structure for mbuf metadata

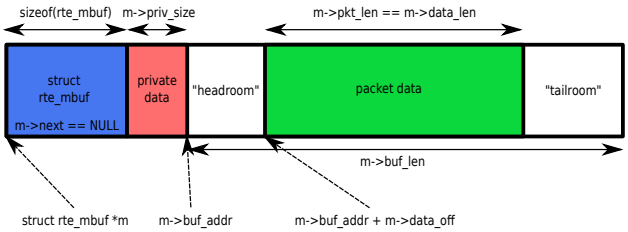


Figure 1. The structure of mbuf for storing packets.

occupies two cache lines. Frequently used items are located in the first cache line. Packet data are stored in the fixed-sized area after the metadata. There is a free space called headroom between the start of buffer for packet data and data itself (Figure 1). The headroom can be used for prepending data to a packet. If a packet is larger than the buffer for packet data mbufs can be chained through their *next* field (Figure 2). The DPDK

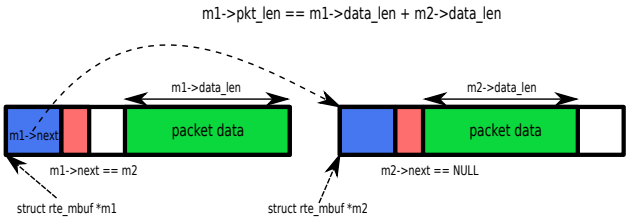


Figure 2. Packet data spread into multiple chained mbufs.

provides a mechanism of so-called *direct* and *indirect* mbufs. An indirect mbuf does not contain actual packet data in its buffer but instead it points to data of its associated direct mbuf. Mbufs are designed so that there is no need to copy memory among buffers. The indirect mbufs can be used to duplicate packets without copying. Packet data are transferred between the host memory and the network card through DMA trans-

fers. The DPDK drivers only fill the DMA descriptors with the physical memory addresses and lengths of the actual packet data. This approach does not allow to fully utilize the bandwidth of the PCI-Express bus as each packet is transferred in a seperate transaction unlike in case of SZE2 interface (described in section 3). However, the main advantage of the single packet per transaction principle is a greater flexibility with the packet manipulation.

The drivers for network devices in the DPDK work in polling mode so that the interrupt overhead is avoided. Therefore they are called Poll Mode Drivers (PMD). The PMD has to implement functions for receiving, transmitting packets, initialization and configuration of device. Besides that, the DPDK Ethernet Device API enables management of a device, filtering, information and statistics displaying. The functions are available through function callbacks exported by the drivers. Drivers do not need to support all functions. The pointers for unsupported functions are set to *NULL* and Ethernet Device API takes care of the rest. Functions for receiving and sending packets transfer bursts of packets to reduce the function call overhead per packet. The size of the burst is adjustable. Note that the higher burst sizes increase the latency. Functions exported by the DPDK drivers are lockless therefore multiple cores cannot poll the same queue from certain port.

3. COMBO cards and SZE2 interface

This section aims on the COMBO network cards [10] and the SZE2 interface.

The COMBO network cards family is based on Field Programmable Gate Array (FPGA) technology. The last generation - COMBOv3 (specifically COMBO-100G and COMBO-80G) cards are connected to the host system with the PCI-Express 3.0 bus. The COMBO network cards are designed for hardware acceleration of network data processing. The FPGA chip on each board allows flexible changing the card's functionality by loading a new firmware to the chip.

The COMBO network cards are managed by a set of proprietary kernel modules divided into multiple layers. The lowest layer implements operations specific for a group of COMBOv3 cards and it is intended to hide differencies between multiple COMBO cards generations. The middle layer kernel module encapsulates the hardware specifics by providing interface for application drivers. The application driver *szedata2* controls DMA transfers among a host system and COMBO cards. User space applications receive and transmit data through the *circular buffer* mapped to user space. The libsze2 library encapsulates the *szedata2* kernel module interface and provides API for user space applications.

From the firmware perspective, the DMA transfers are managed by a DMA controller located in the FPGA firmware and a pair of buffers (one in the host system and one in the network card). Each buffer has a start-of-data and an end-of-data pointer. The DMA controller manages the transfers based on values of these pointers. Each direction is associated to one or more dedicated DMA channels. Each channel has its own buffers. The number of channels varies among different firmwares.

The size of the *circular buffer* in the host system memory reaches dozens of megabytes. The buffer (Figure 3) is composed of several memory blocks aligned to 4 kB boundaries which take multiple memory pages (referred as *data blocks* in the further text). Each *data block* is described by a 64-bit descriptor which contains an upper non-zero part of the address of the *data block* and a number of pages in the block. The descriptors are grouped into a dedicated memory block (referred as *descriptors block*). The last descriptor located in the *descriptors block* does not point to the *data block* but to the start of the *descriptors block*. The

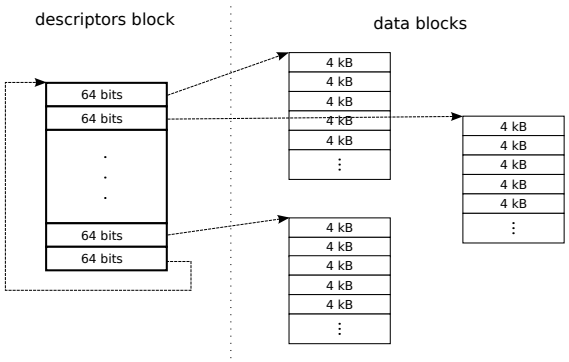


Figure 3. SZE2 *circular buffer* is composed of data blocks and block with descriptors.

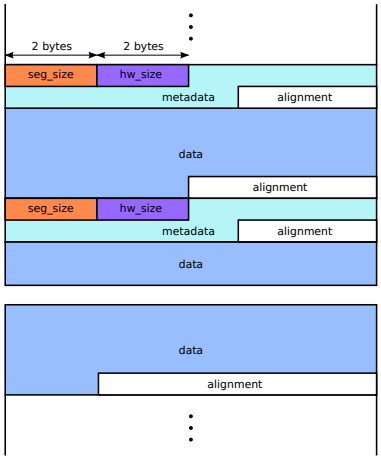


Figure 4. Detail of a SZE2 segment placed at the border of two data blocks.

descriptor pointing to the *descriptors block* is distinguished from descriptors pointing to the *data blocks* by the least significant bit.

The SZE2 interface was originally designed universally to transfer different types of data through the PCI-Express. In the other words, it is not limited only to the network packets. The SZE2 interface works with segments of data that have variable size consisting of a header and data. The header contains a size of the whole segment including header, size of the header part and metadata related to an accelerating core. Both the header and data parts are aligned to 8 bytes (Figure 4).

4. Poll Mode Driver szedata2

The szedata2 PMD encapsulates the SZE2 interface for the DPDK environment.

The initial version of the szedata2 PMD supports only those functions which are necessary for starting, stopping the device and functions for reception and transmission.

Data transferred through the PCI-Express bus are divided into transaction layer packets (TLP) which consist of a header and a payload. The maximal size of the payload for a single TLP is 256 B. A better utilization of the bus bandwidth is achieved by sending longer payloads at once.

Many original DPDK drivers transfer packets using descriptors. Each descriptor is associated with a physical address of an mbuf's payload and can transfer a single packet (except jumbo frames that are scattered around multiple mbufs). This approach is inconvenient for short packets as it increases the overhead of the PCI-Express TLPs as the TLPs with shorter payloads are used.

The SZE2 interface does not use descriptors to transfer separate packets through DMA like other network cards. SZE2 segments are placed continuously one by one into buffer composed of large blocks. Such organization allows to reduce the overhead of the PCI-Express TLPs, especially for short packets by aggregating them in common TLPs with longer payloads. On the other hand, the SZE2 approach is suitable only for a specific class of applications. The SZE2 approach complicates the situation for applications that defer processing of certain packets. This can lead to a higher memory fragmentation as there might be only few packets in a single memory block which need a deferred processing. Such packets block to reuse the whole data block full of other uninteresting packets. Therefore to support DPDK applications without restrictions, the received packets are copied from the

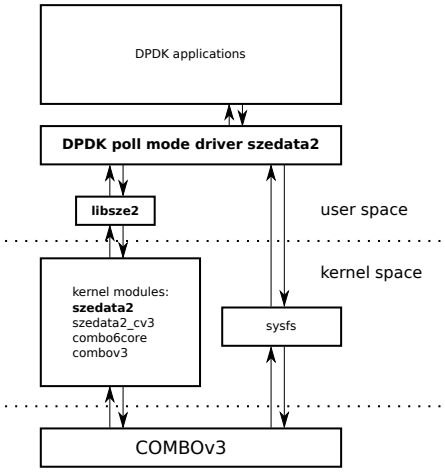


Figure 5. Structure of DPDK applications built over COMBOv3 network card.

SZE2 *circular buffer* into mbufs and packets that are going to be transmitted are copied in the opposite direction.

The initial version of the szedata2 PMD has become a part of the DPDK mainline since the release 2.2.0. The code is located in */drivers/net/szedata2* directory of the DPDK tree. The DPDK release 2.2.0 with the szedata2 PMD code can be downloaded from the Download section of the DPDK project website[1]. The code from the current git master branch is also accessible for online browsing at the DPDK project website[2]. The release 2.2.0 contains also documentation for the szedata2 PMD as a part of the DPDK Network Interface Controller Drivers Guide [11].

The next goal of my work might be adding a support for the card configuration and management. The network card address space is mapped to the memory of the PMD. An access to the card address space allows implementation of configuration and control functions. Currently, the functions for setting MAC address check mode, getting link status, setting link up and down are supported. The functions for setting MAC addresses, maximum transmission unit, reading hardware statistics and others are planned for a next version.

5. Benchmarks and Results

The szedata2 PMD has been developed and benchmarked on top of the DPDK version 2.1.0. The configuration of the system was:

- a host machine equipped with one COMBO-100G card (information is summarized in Table 1)

[1] http://dpdk.org/download

[2] http://dpdk.org/browse/dpdk/tree/drivers/net/szedata2

Table 1. Host Machine Information

CPUs	2x Xeon(R) CPU E5-2660 v3 @ 2.60GHz
Cores	2x10 (2x20) - enabled Hyper-Threading
OS	Scientific Linux release 6.5 (Carbon)
Kernel	2.6.32-431.1.2.el6.x86_64

- Spirent Testcenter hardware tester used as packet generator
- the host machine connected to the packet generator as displayed on Figure 6

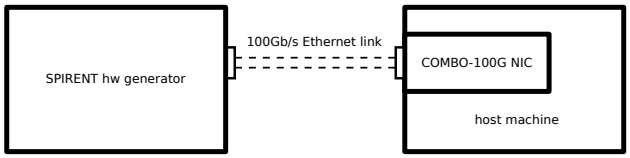


Figure 6. Illustration of benchmark setup.

- the DPDK application *testpmd* ran on the host machine
- COMBO kernel modules version 0.9.2
- libsze2 version 1.1.4
- DPDK version 2.1.0
- COMBO-100G firmware NIC_100G1_LR4 with 8 receiving and 8 transmitting channels

Packet rates for the following cases have been measured:

1. Reception of unidirectional traffic - *testpmd* was run with option *--forward-mode=rxonly*.
2. Transmission of unidirectional traffic - *testpmd* was run with option *--forward-mode=txonly*.
3. Bidirectional traffic forwarding (received data was transmitted back through the same port) - *testpmd* was run with option *--forward-mode=io*.

For each case the following configurations have been measured:

1. 1 forwarding queue running on 1 physical core, 1 used logical core per physical core (label in graphs 1C/1T)
2. 2 forwarding queues running on 1 physical core, 2 used logical cores per physical core (label in graphs 1C/2T)
3. 2 forwarding queues running on 2 physical cores, 1 used logical core per physical core (label in graphs 2C/1T)
4. 4 forwarding queues running on 2 physical cores, 2 used logical cores per physical core (label in graphs 2C/2T)
5. 4 forwarding queues running on 4 physical cores, 1 used logical core per physical core (label in graphs 4C/1T)
6. 8 forwarding queues running on 4 physical cores, 2 used logical cores per physical core (label in graphs 4C/2T)
7. 8 forwarding queues running on 8 physical cores, 1 used logical core per physical core (label in graphs 8C/1T)

Frame sizes 64, 128, 256, 512, 1024, 1280, 1518 as suggested by RFC 2544 [12] have been used for each configuration. The statistics for the reception case were counted using *ibufctl* application[3]. The counters from the Spirent Testcenter were used for the transmission and forwarding cases.

Figure 7 displays measured results of packet rates for all stated cases and configurations.

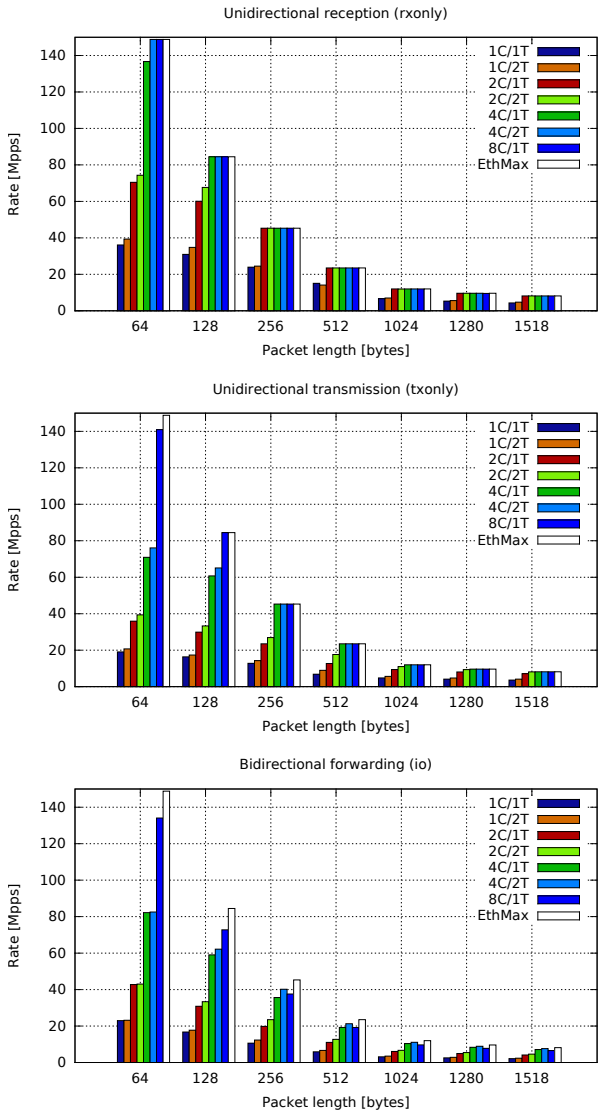


Figure 7. Packet rates for unidirectional reception, transmission and bidirectional forwarding. A column *EthMax* represents theoretical maximum value for the 100G Ethernet.

[3] application for accessing the registers from the COMBO card's IBUF component

The data reception at 100 Gb/s wirespeed has been achieved with 8 CPU cores for 64 B long frames. 4 cores have been needed for 128 B long frames and from 256 B the wirespeed has been achieved with 2 cores.

The data transmission at 100 Gb/s wirespeed has been achieved with 8 CPU cores for 128 B long frames. From the 256 B long frames, only 4 cores have been sufficient.

A bidirectional traffic can be handled at more than 90% rates of 100 Gb/s wirespeed with 8 CPU cores even for 64 B long frames.

6. Conclusions

This paper describes implementation of the DPDK szedata2 PMD which connects the SZE2 interface of COMBO network cards through the libsze2 library to the DPDK. The driver has become a part of the DPDK mainline since the release 2.2.0. Packets transferred through the SZE2 interface are aggregated into continuous buffers to reduce the number of required PCI-Express transactions. Due to this fact, the data has to be copied from SZE2 buffers into DPDK mbufs resulting in a higher load of the host's memory subsystem.

The set of benchmarks has been performed to measure the performance of the szedata2 PMD. The network card COMBO-100G can handle 64 B long frames using multiple CPU cores at rates:

- over 140 Mpps in unidirectional reception and transmission
- up to 134 Mpps in bidirectional traffic

This paper shows that DPDK applications can receive and transmit data through COMBO-100G card at 100 Gb/s Ethernet wirespeed.

Further work is needed to implement more management and configuration functions for COMBO cards in the DPDK driver. Currently, the szedata2 PMD depends on kernel modules for COMBO cards and libsze2 library. In the future, the kernel modules functionality could be integrated into the szedata2 PMD and the proprietary dependencies could be removed.

Acknowledgements

I would like to thank Ing. Martin Špinler, Ing. Viktor Puš, Ph.D. from CESNET for the cooperation and precious advices and to my supervisor Ing. Jan Viktorin for his guidance.

References

[1] Data plane development kit. http://www.dpdk.org.

[2] Netmap - the fast packet i/o framework. http://info.iet.unipi.it/~luigi/netmap/.

[3] Pf_ring - ntop. http://www.ntop.org/products/packet-capture/pf_ring/.

[4] Openonload. http://www.openonload.org/.

[5] Sebastian Gallenmüller, Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. Comparison of frameworks for high-performance packet io. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, pages 29–38, Washington, DC, USA, 2015. IEEE Computer Society.

[6] Tom Barbette, Cyril Soldani, and Laurent Mathy. Fast userspace packet processing. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, pages 5–16, Washington, DC, USA, 2015. IEEE Computer Society.

[7] Jiří SLABÝ. Rapid data transfers on combo platform [online]. Master's thesis, Masaryk University, Faculty of Informatics, Brno, 2008 [cit. 2016-04-04]. http://is.muni.cz/th/98734/fi_m/.

[8] Andrej HANK. Design of network applications for a netcope platform [online]. Master's thesis, Brno University of Technology, Faculty of Information Technology, Brno, 2009 [cit. 2016-04-04]. http://www.fit.vutbr.cz/study/DP/DP.php.cs?id=8195.

[9] Dpdk programmer's guide release 2.2.0, 2016 [cit. 2016-04-04]. http://dpdk.org/doc/pdf-guides/prog_guide-2.2.pdf.

[10] Project liberouter website - cards. https://www.liberouter.org/technologies/cards/.

[11] Dpdk network interface controller drivers release 2.2.0, 2016 [cit. 2016-04-04]. http://dpdk.org/doc/pdf-guides/nics-2.2.pdf.

[12] S. Bradner and J. McQuaid. Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), March 1999. Updated by RFCs 6201, 6815.