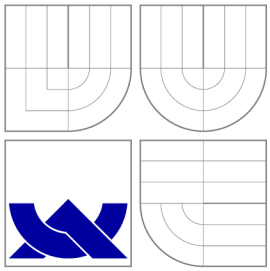
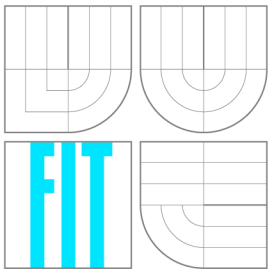


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

DPDK NAD SÍŤOVÝMI KARTAMI COMBO

DPDK FOR COMBO NETWORK CARDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MATEJ VIDO

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAN VIKTORIN

BRNO 2016

Zadání bakalářské práce/18312/2015/xvidom00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Vido Matej**
Obor: Informační technologie
Téma: **DPDK nad síťovými kartami COMBO**
DPDK for COMBO Network Cards
Kategorie: Počítačová architektura

Pokyny:

1. Seznamte se s knihovnou DPDK, s kartami rodiny COMBO a knihovnou libsze2 pro rychlé datové přenosy.
2. Proveďte srovnání principů paketových přenosů v systémech libsze2 a DPDK navrhněte vhodný způsob jejich propojení.
3. Implementujte podporu karet COMBO v knihovně DPDK a otestujte na zvolené síťové aplikaci.
4. Změřte a vyhodnoťte výkonnost výsledného řešení.
5. Diskutujte možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Viktorin Jan, Ing.**, UPSY FIT VUT
Konzultant: Puš Viktor, Ing., Ph.D., CESNET
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

Softvérový aplikačný rámec Data Plane Development Kit poskytuje štandardné rozhranie pre rýchle spracovanie paketov v užívateľskom priestore. DPDK podporuje sieťové zariadenia od viacerých výrobcov a rôzne architektúry. Združenie CESNET vyvíja sieťové karty rodiny COMBO pre Ethernet o rýchlostiach do 100 Gb/s. Prenos dát medzi sieťovými kartami COMBO a hostiteľským systémom je zabezpečovaný rozhraním SZE2. Táto práca popisuje návrh pridania podpory sieťových kariet COMBO do DPDK pomocou implementácie ovládača pre DPDK nazvaného szedata2. Vytvorený ovládač sa stal súčasťou DPDK od verzie 2.2.0 (december 2015). V práci sú ďalej popísané prevedené merania výkonnosti a dosiahnuté výsledky. Pri meraniach sa podarilo prijímať a vysielať dáta plnou rýchlosťou linky o rýchlosti 100 Gb/s.

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 Ethernet traffic up to 100 Gb/s through their SZE2 interface. This thesis describes the design and implementation of the DPDK user space driver for COMBO network cards. The driver is called szedata2 and has already become a part of the DPDK mainline in the version 2.2.0 (December 2015). The thesis describes also the measurements and the accomplished results. Packets have been received and transmitted at the wirespeed of the 100 Gb/s link.

Kľúčové slová

DPDK, Data Plane Development Kit, sieťové karty COMBO, COMBO-100G, rozhranie SZE2, knižnica libsze2, szedata2, 100 Gb/s

Keywords

DPDK, Data Plane Development Kit, COMBO network cards, COMBO-100G, SZE2 interface, libsze2 library, szedata2, 100 Gb/s

Citácia

VIDO, Matej. *DPDK nad síťovými kartami COMBO*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Viktorin Jan.

DPDK nad síťovými kartami COMBO

Prehlásenie

Vyhlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jana Viktorina. Ďalšie informácie mi poskytol konzultant Ing. Viktor Puš, Ph.D. zo združenia CESNET. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

. . . . . .
Matej Vido
16. mája 2016

Poďakovanie

Rád by som poďakoval vedúcemu Ing. Janovi Viktorinovi za venovaný čas a odbornú pomoc pri vypracovaní práce. Ďalej by som chcel poďakovať Ing. Viktorovi Pušovi, Ph.D. a Ing. Martinovi Špinlerovi zo združenia CESNET za poskytnuté konzultácie a prístup k potrebnému technickému vybaveniu. V neposlednom rade ďakujem rodine za podporu pri štúdiu.

Obsah

Kapitola 1

Úvod

Sledovanie a analýza sieťových dát patrí k dôležitým súčastiam správy počítačových sietí. Medzi bežné prostriedky pre sieťovú komunikáciu pod operačnými systémami založenými na Linuxe patria schránky a knižnica *libpcap*[1]. Schránky tvoria jednotné rozhranie medzi aplikáciami bežiacimi v užívateľskom priestore a sieťovým protokolovým zásobníkom v jadre. Knižnica *libpcap* poskytuje API[2] pre zachytávanie paketov na spojovej vrstve.

V roku 2010 bol definovaný štandard IEEE 802.3ba-2010 pre Ethernet o rýchlostiach 40 Gb/s a 100 Gb/s. Najkratší rámec Ethernetu na spojovej vrstve má dĺžku 64 B vrátane kontrolného súčtu. Na fyzickej vrstve zaberá takýto paket 84 B (oproti spojovej vrstve navyše 7 B preambula, 1 B oddeľovač začiatku rámca a 12 B medzera medzi paketmi). V sieťach s Ethernetom o rýchlosti 100 Gb/s je možné preniesť až 148,8 miliónov najkratších paketov za sekundu. Aby bolo možné zachytávať a analyzovať dáta pri plnej rýchlosti linky, musí procesor spracovať jeden paket za 6,72 ns (t. j. 20 cyklov pri frekvencii procesora 3 GHz).

Sieťový protokolový zásobník v jadre operačného systému je navrhnutý všeobecne s ohľadom na veľkú množinu sieťových protokolov, širokú funkcionalitu a kompatibilitu. So zvyšovaním rýchlostí sieťových prenosových štandardov prestávajú bežné prostriedky využívajúce sieťový zásobník v jadre stačiť. Z tohto dôvodu vznikajú hardvérové aj softvérové riešenia pre spracovávanie vysokorýchlostných sieťových tokov. Zariadenia založené na technológii FPGA (*Field Programmable Gate Array* – programovateľné hradlové pole), sieťové procesory, či systémy využívajúce grafické procesory (GPU) umožňujú presunúť záťaž do hardvéru a odľahčiť tak hlavný procesor (CPU).

Softvérové aplikačné rámce a knižnice pre rýchle spracovanie paketov v užívateľskom priestore obchádzajú jadro operačného systému a vyhýbajú sa tak výkonovým nedostatkom jeho sieťového zásobníku. Data Plane Development Kit [13] (DPDK) patrí medzi tieto moderné aplikačné rámce. DPDK obsahuje sadu ovládačov pre sieťové karty od rôznych výrobcov. Ďalšími podobnými softvérovými riešeniami sú napríklad netmap [16] a PF_RING ZC [18].

Združenie CESNET vyvíja v rámci projektu Liberouter sieťové karty rodiny COMBO pre Ethernet o rýchlostiach 10 Gb/s, 40 Gb/s a 100 Gb/s. Karty COMBO sú založené na technológii čipov FPGA, vďaka čomu umožňujú spracovávanie dát aj priamo v karte bez účasti procesoru. Rýchly prenos dát zo sieťových kariet rodiny COMBO pre spracovanie v softvéri je zabezpečený rozhraním SZE2[3]. Rozhranie SZE2 používa DMA (*Direct Memory*

[1] z anglického *Packet Capture Library*

[2] *Application Programming Interface* – rozhranie pre programovanie aplikácií

[3] *Straight ZEro copy*

Access – priamy prístup do pamäte) prenosy cez zbernicu PCI-Express. Toto rozhranie je aplikáciám v užívateľskom priestore sprístupnené pomocou knižnice *libsze2*. Rozhranie SZE2 a knižnica *libsze2* boli predstavené a navrhnuté v diplomových prácach Jiřího Slabého [10] a Andreja Hanka [6].

Táto práca sa zaoberá princípom napojenia rozhrania SZE2 sieťových kariet rodiny COMBO na aplikačný rámec DPDK. Prepojenie týchto systémov bolo dosiahnuté implementáciou DPDK ovládača nazvaného *szedata2* bežiaceho v užívateľskom priestore. DPDK ovládač *szedata2* umožňuje prijímať a odosielať dáta cez sieťové karty COMBO s použitím rozšíreného API a zároveň dosahovať vysoké rýchlosti prenosov.

Kapitola 2 na začiatku predstavuje výkonnostné nedostatky linuxového sieťového zásobníka a techniky umožňujúce efektívnejšie spracovanie sieťových dát. Ďalej poskytuje základné porovnanie aplikačných rámcov obchádzajúcich jadro operačného systému, popis rozhrania schránok a knižnice *libpcap*. Na konci je podrobnejšie rozobratá architektúra DPDK, charakteristika sieťových kariet rodiny COMBO a princíp prenosov dát cez rozhranie SZE2. Kapitola 3 popisuje návrh a princíp prepojenia rozhrania SZE2 a aplikačného rámca DPDK. V kapitole 4 je popísaná implementácia DPDK ovládača *szedata2*. Popis meraní, dosiahnuté výsledky a zhodnotenie výkonnosti DPDK ovládača *szedata2* sú v kapitole 5.

Kapitola 2

Sieťové rozhrania v operačných systémoch založených na Linuxe

V tejto kapitole sú popísané rozhrania a knižnice, ktoré umožňujú prijímať a odosielať dáta cez sieť v operačných systémoch založených na Linuxe. Na začiatku, v sekcii 2.1, je popísaná architektúra linuxového sieťového zásobníka a ovládačov sieťových zariadení, ktoré tvoria vrstvu medzi sieťovou kartou a rozhraním schránok (sekcia 2.2), resp. knižnicou *libpcap* (sekcia 2.3). Ďalej sú v tejto sekcii uvedené výkonové nedostatky, ktoré podnietili vznik softvérových aplikačných rámcov (sekcia 2.5) obchádzajúcich jadro operačného systému. V sekcii 2.4 sú uvedené techniky použité na prekonanie výkonových obmedzení sieťového zásobníka. Aplikačnému rámcu DPDK je venovaná sekcia 2.6. Sekcia 2.7 popisuje architektúru a princíp rýchlych dátových prenosov cez rozhranie SZE2 medzi sieťovými kartami rodiny COMBO a hostiteľským systémom. V tejto sekcii je taktiež popísaná knižnica *libsze2*, ktorá bola vytvorená za účelom zapúzdrenia rozhrania SZE2 pre užívateľské aplikácie.

2.1 Sieťový protokolový zásobník v Linuxe

Podpora pripojenia počítača do siete zo strany operačného systému bola navrhnutá tak, aby spĺňala všeobecné požiadavky zo strany sieťových kariet a protokolov. Jadro systému poskytuje kompletný protokolový zásobník, cez ktorý putujú pakety medzi ovládačom sieťového zariadenia a užívateľským priestorom.

Príjem a odosielanie paketov v jadre je riadené na princípe prerušení. Každý paket prijatý sieťovou kartou je priradený k deskriptoru v prijímacej (RX[1]) fronte sieťovej karty. Deskriptor obsahuje informácie o adrese pamäti, kam je následne paket prenesený DMA prenosmi. Po prenesení každého paketu je vyvolané prerušenie, ktorého obslužná rutina zabezpečí skopírovanie paketu do štruktúry `sk_buff` v bufferi jadra. Pakety v `sk_buff` štruktúre následne putujú cez sieťový zásobník až do užívateľského priestoru. Vo vysielacom (TX[2]) smere je prerušenie vyvolané po úspešnom DMA prenose paketu do TX fronty v karte informujúc o možnosti prenosu ďalšieho paketu. Nevýhodou tohto prístupu je potreba vyvolať prerušenie pre každý prenášaný paket, kvôli čomu dochádza k nadmernému zaťaženiu systému v prípade vysokej sieťovej prevádzky [5].

[1] z anglického *reception*
[2] z anglického *transmission*

Zlepšenie výkonu sieťového zásobníku prinieslo začlenenie NAPI[3] do jadra od verzie 2.6 [5]. NAPI využíva dva princípy:

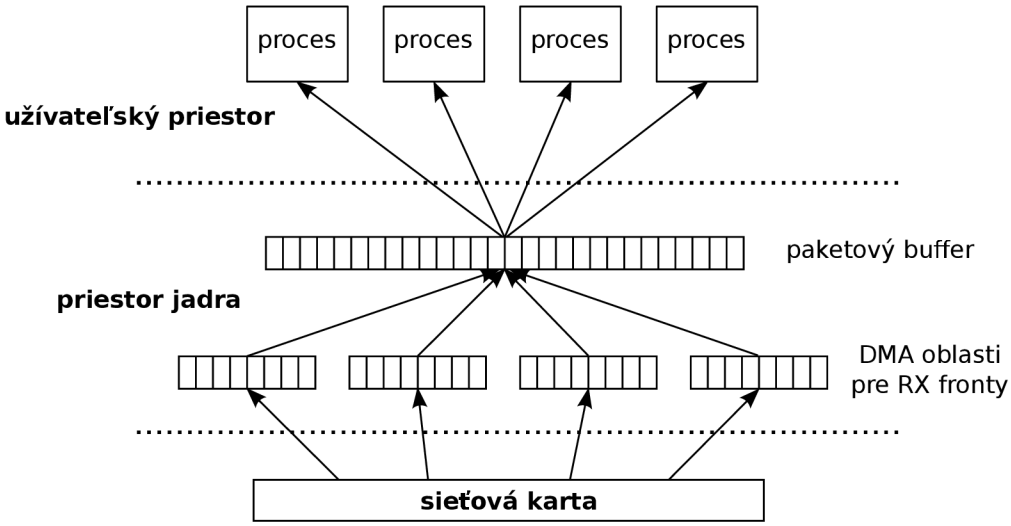
1. **Zníženie počtu prerušení.** Keď je vyvolané prerušenie, NAPI umožňuje zakázať na určitý čas prerušenia. Ak je dosupných viac paketov, ktoré je možné skopírovať medzi bufferom jadra a pamäťou pre DMA prenosy, skopírujú sa všetky pakety až kým nie sú ďalšie pakety dostupné a prerušenie sa znovu povolí [15, 5].

2. **Zahadzovanie paketov.** Keď dôjde k zahlteniu systému, je nutné zahadzovať pakety. Ovládače podporujúce NAPI umožňujú zahadzovanie paketov v sieťovej karte, bez toho aby sa dostali do jadra [15, 5].

Použitie NAPI prinieslo zníženie réžie spôsobenej obsluhou prerušení. Toto však nie je dostačujúce pre zvládnutie spracovania tokov dát pri Ethernete o rýchlostiach 10 Gb/s, 40 Gb/s a 100 Gb/s, pretože jadro obsahuje ďalšie architektonické problémy zvyšujúce réžiu. Podľa Garcia-Dorado et al. [5] sú významné nasledujúce nedostatky:

1. **Alokovanie a uvoľňovanie zdrojov.** Pre každý paket je zvlášť alokovaný deskriptor, do ktorého sa ukladajú informácie potrebné pre DMA prenos, po prenose paketu je deskriptor znova uvoľnený. Ďalšiu réžiu predstavuje aj alokovanie a uvoľňovanie štrukúry `sk_buff`, ktorá nesie sieťovým zásobníkom informácie o pakete.

2. **Serializovaný prístup k dátam.** Moderné sieťové karty umožňujú rozdelenie toku dát do viacerých RX a TX front. Vďaka tomuto rozdeleniu je možné efektívnejšie využívať viacjadrové procesory. Každá RX/TX fronta môže byť spracovávaná samostatným jadrom. Problém je, že sieťový zásobník spája pakety zo všetkých front do jedného bufferu v jadre kvôli analýze na sieťovej a transportnej vrstve. Užívateľské procesy tak nie sú schopné prijímať dáta z jednotlivých front. Tento proces je znázornený na obrázku 2.1.



Obr. 2.1: Znázornenie spracovania viacerých front zo sieťovej karty v jadre [3, 9, 5].

[3] *New API*

3. **Viacnásobné kopírovanie dát.** Každý paket je kopírovaný z oblasti pre DMA prenosy do bufferu v jadre a následne z bufferu v jadre do aplikácie v užívateľskom priestore.

4. **Prepínanie kontextu medzi užívateľským priestorom a jadrom.** Užívateľská aplikácia musí pre prijatie každého paketu vykonať systémové volanie, ktoré vyžaduje prepnutie kontextu.

2.2 Schránky

Schránky (*sockets*) predstavujú základný nástroj pre programovanie sieťových aplikácií. Tvoria jednotné rozhranie medzi procesmi v užívateľskom priestore a sieťovým protokolovým zásobníkom v jadre. Protokolové moduly sú zoskupené do rodín protokolov a typov schránok. Rozhranie schránok bolo prvýkrát implementované v operačnom systéme unixového typu BSD[4] verzie 4.2BSD v roku 1983. Mnoho unixových systémov založilo svoju implementáciu na sieťových kódoch a API schránok z BSD. Linux medzi tieto systémy nepatrí. Jeho implementácia sieťových kódov a schránok bola vyvíjaná od základov, nie je odvodená od BSD kódov [11].

Prístup k transportnej vrstve Komunikácia na úrovni transportnej vrstvy je postavená na schránkach protokolovej rodiny `AF_INET` pre sieťový protokol IPv4 a `AF_INET6` pre sieťový protokol IPv6. TCP schránky sú typu `SOCK_STREAM`. Informácie o TCP schránkach je možné nájsť v manuálových stránkach *tcp(7)* [22]. UDP schránky sú typu `SOCK_DGRAM`. UDP schránkam sú venované manuálové stránky *udp(7)* [23].

Prístup k sieťovej vrstve Pre komunikáciu na úrovni sieťovej vrstvy je možné vytvoriť tzv. *raw* schránky pomocou typu `SOCK_RAW`. Používajú sa protokolové rodiny `AF_INET` pre IPv4 a `AF_INET6` pre IPv6. Schránky tohto typu umožňujú príjem a odosielanie dát protokolov ICMP, IGMP a IP. *Raw* schránky môže vytvárať iba privilegovaný proces[5] alebo proces s nastavenou schopnosťou `CAP_NET_RAW` [21].

Prístup k spojovej vrstve V užívateľskom priestore operačných systémov založených na Linuxe je možné zachytávať a odosielať dáta na úrovni spojovej vrstvy pomocou schránok protokolovej rodiny `AF_PACKET`. Okrem schránok tejto protokolovej rodiny je možné použiť aj knižnicu libpcap, ktorá je popísaná v sekcii 2.3. Rodina schránok `AF_PACKET` umožňuje tvorbu protokolových modulov nad fyzickou vrstvou v užívateľskom priestore. Je možné vytvoriť dva typy `AF_PACKET` schránok:

- `SOCK_RAW` – kompletné pakety spojovej vrstvy
- `SOCK_DGRAM` – tzv. „*cooked*“ pakety, t. j. pakety s odstránenou hlavičkou spojovej vrstvy

Vo východzom stave sú predávané `AF_PACKET` schránke pakety zo všetkých rozhraní. Pomocou volania funkcie `bind()` je možné špecifikovať rozhranie, z ktorého sa majú prijímať pakety do schránky. `AF_PACKET` schránky môžu vytvárať len procesy s nastavenou schopnosťou `CAP_NET_RAW` [17, 11].

[4] Berkeley Software Distribution
[5] proces s efektívnym UID (*user ID*) 0

2.3 Knižnica libpcap

Okrem `AF_PACKET` schránok je knižnica *libpcap* ďalším rozhraním aplikácií v užívateľskom priestore pre manipuláciu s dátami na spojovej vrstve. Pre operačné systémy Windows existuje alternatíva v podobe knižnice WinPcap.

Knižnica libpcap je prenositeľná knižnica pre programovacie jazyky C a C++, ktorá bola primárne navrhnutá na zachytávanie sieťovej prevádzky. Okrem zachytávania dát knižnica umožňuje aj odosielanie dát na úrovni spojovej vrstvy, ukladanie zachytených dát do súboru a čítanie dát zo súboru. Knižnica poskytuje jednotné rozhranie pre užívateľský priestor nezávislé na operačnom systéme. Zapúzdruje konkrétny mechanizmus zachytávania paketov poskytovaný systémom, či už ide o BPF[6] na BSD systémoch, DLPI[7] na HP-UX a Solaris systémoch alebo `AF_PACKET` schránky v Linuxe.

Zatiaľ čo čítanie dát zo súboru nevyžaduje špeciálne oprávnenia, zachytávanie dát zo sieťového rozhrania môže vyžadovať určité privilégiá. Podrobné informácie o požadovaných privilégiách na konkrétnych operačných systémoch je možné nájsť v manuálových stránkach [7].

2.4 Techniky pre zvýšenie výkonnosti spracovania dát zo siete

V sekcii 2.1 boli popísané nedostatky sieťového protokolového zásobníka v jadre. Táto sekcia popisuje spoločné vlastnosti a techniky, ktoré používajú moderné aplikačné rámce pre vysokorýchlostné spracovanie sieťových dát. Niektoré významné aplikačné rámce (okrem DPDK, ktorému je venovaná samostatná sekcia) sú popísané v sekcii 2.5. Tieto aplikačné rámce poskytujú prevažne základnú vstupno-výstupnú funkcionalitu. Vyššie vrstvy ako sieťová a transportná vrstva, musia byť implementované užívateľskou aplikáciou. Nasledujúce techniky umožňujú zvýšenie výkonnosti spracovania sieťových dát:

- **Vlastná správa pamäťových zdrojov.** Buffere pre ukladanie paketov sú alokované pred začiatkom spustenia dátových prenosov. Pri odoslaní paketu nie sú buffere uvoľňované späť systému, ale sú spravované interne konkrétnym aplikačným rámcom a znovu použité pre prijatie ďalších paketov. Štruktúry pre ukladanie paketov môžu byť zjednodušené oproti `sk_buff` štruktúre.
- **Nulové kopírovanie dát.** Buffere, do ktorých sú prenášané dáta z karty DMA prenosmi sú namapované do užívateľského priestoru. Aplikácia tak môže pracovať priamo s prijatými paketmi bez nutnosti kopírovať dáta. Niektoré prístupy vyžadujú samostatné buffere pre RX a TX smer. V prípade, že sú dáta preposielané zo vstupu na výstup je tak nutné kopírovať dáta medzi vstupným a výstupným bufferom.
- **Spracovanie zhlukov paketov** umožňuje rozložiť réžiu spojenú s prístupom ku karte medzi viac paketov, prípadne využiť špeciálne vlastnosti kariet pre hromadné prijímanie alebo odosielanie dát.
- **Obrovské stránky[8].** Obrovské stránky zvyšujú úspešnosť vyhľadávania v TLB[9] vďaka čomu je znížená réžia prekladu virtuálnych adries na fyzické.

[6] *Berkeley Packet Filter*
[7] *Data Link Provider Interface*
[8] *huge pages*
[9] *Translation Lookaside Buffer*

- **Paralelné spracovanie.** Viaceré RX a TX fronty sú vďaka mapovaniu do užívateľského priestoru a obchádzaniu bufferov v jadre priamo dostupné z užívateľskej aplikácie, ktorá môže tak naplno využiť viacjadrové procesory.
- **Obchádzanie sieťového zásobníka v jadre.** Vďaka priamemu mapovaniu bufferov s dátami z karty do užívateľského priestoru je možné úplne obchádzať protokolový zásobník v jadre, ktorý je kvôli jeho všeobecnej funkcionalite príliš pomalý. Okrem toho je tak možné ušetriť aj réžiu spojenú so systémovými volaniami. Na druhú stranu je ale nutné postaviť vlastný sieťový zásobník v užívateľskej aplikácii. Avšak takýto zásobník môže byť optimalizovaný pre konkrétny účel.
- ***Polling***[10] umožňuje odstrániť réžiu spojenú s obsluhou prerušení. Na druhú stranu prináša väčšie zaťaženie procesoru.
- **Afinita.** Rozlišuje sa procesorová a pamäťová afinita. Procesorová afinita je technika umožňujúca pripútanie vykonávania vlákna alebo procesu na určité procesorové jadro. Pamäťová afinita znamená to, že vlákno alebo proces musí používať kusy pamäte priradené procesoru, na ktorom beží [5, 12].
- ***Prefetching*** je technológia umožňujúca načítať do cache pamäte procesoru dáta, ktoré sa budú spracovávať neskôr, počas spracovávania predchádzajúcich dát. Vhodné použitie tejto techniky dokáže znížiť réžiu spojenú s prekladom virtuálnych adries na fyzické.

2.5 Softvérové riešenia obchádzajúce sieťový zásobník v jadre

V tejto sekcii sú predstavené základné charakteristiky aplikačných rámcov *PF_RING ZC* a *netmap*. Tieto aplikačné rámce spolu s DPDK sú podrobnejšie popísané a porovnávané v literatúre [4, 1, 12, 5].

Netmap Netmap sprístupňuje paketové buffere aplikáciám pomocou mapovania do užívateľského priestoru. Podporuje paralelné spracovanie vo viacerých vstupno-výstupných frontách a v zhlukoch paketov. Pamäťové zdroje sú predalokované počas inicializácie. Netmap používa štandardné systémové volania ako `ioctl()`, `poll()`, `select()` pre iniciovanie prenosov. Tieto volania iba aktualizujú paketové buffere a kontrolujú platnosť dát z užívateľskej aplikácie. Ovládače pre netmap sú založené na bežných linuxových ovládačoch. Keď nie je spustená aplikácia pre netmap, ovládač poskytuje dáta štandardnému sieťovému zásobníku v jadre. Pri spustení aplikácie založenej na netmape je ovládač prepnutý do špeciálneho režimu, v ktorom presmerováva dáta do štruktúr špecifických pre netmap a pakety nie sú ďalej dostupné bežným sieťovým prostriedkom operačného systému [4, 5]. Netmap podporuje tiež napojenie na *libpcap*. V súčasnej dobe je netmap dostupný pre Linux, FreeBSD aj Windows [16].

PF_RING ZC PF_RING ZC[11] nepoužíva bežné systémové volania, ale poskytuje svoje vlastné API, ktoré umožňuje pohodlné viacjadrové spracovanie. Ako naznačuje názov, dôležitým princípom je nulové kopírovanie paketov. PF_RING je postavený s ohľadom na

[10] programová obsluha zariadenia, aplikácia synchrónne zisťuje stav zariadenia
[11] ZC je skratka pre *zero copy* – nulové kopírovanie

NUMA[12] architektúru. Ovládač je založený na bežnom linuxovom ovládači a správa sa transparentne pre operačný systém (podobne ako ovládač pre netmap), ak nie je spustená žiadna aplikácia založená na PF_RING [18, 4]. PF_RING umožňuje posunúť pakety zo svojho rozhrania aj do sieťového zásobníku v jadre, ale treba počítať so zníženou výkonnosťou. PF_RING je dostupný pre Linux.

2.6 Data Plane Development Kit

DPDK je projekt s otvorenými zdrojovými kódmi pod BSD licenciou s výnimkou modulov jadra, ktoré sú pod licenciou GNU GPL[13]. V súčasnosti sú hlavnými správcami projektu spoločnosti 6WIND a Intel. DPDK obsahuje sadu knižníc a ovládačov určených na rýchle spracovanie paketov v užívateľskom priestore. Súčasťou sú aj príklady na tvorbu aplikácií využívajúcich DPDK knižnice. DPDK bolo navrhnuté tak, aby mohlo bežať na rôznych architektúrach procesorov. Medzi podporované architektúry momentálne patrí Intel x86, IBM Power 8, EZchip TILE-Gx a ARM [13]. DPDK je určené predovšetkým pre operačný systém Linux avšak časť knižníc je prispôsobená aj pre operačný systém FreeBSD. DPDK obsahuje ovládače pre sieťové karty od viacerých výrobcov, medzi ktorých patria Chelsio, Cisco, Emulex, Intel, Mellanox, Netronome a QLogic. Táto práca popisuje návrh, implementáciu a experimenty s ovládačom *szedata2* pre sieťové karty COMBO-100G, ktoré vyvíja združenie CESNET. Tento ovládač sa stal súčasťou DPDK od verzie 2.2.0, ktorá vyšla v decembri 2015.

DPDK vytvára abstraktnú vrstvu nazvanú EAL[14], ktorá zapúzdruje osobitosti architektúry, prekladača a konkrétnej platformy. Nad vrstvou EAL sú potom postavené ostatné knižnice, ktoré môžu byť nezávislé od konkrétneho prostredia. Vrstva EAL pre konkrétne prostredie sa prispôsobuje pomocou konfiguračných súborov. V nasledujúcich oddieloch je popísaná vrstva EAL a knižnice dôležité z pohľadu tvorby ovládača pre sieťové zariadenie.

2.6.1 Vývojový proces DPDK

Vývoj prebieha prostredníctvom vývojárskej emailovej konferencie. Vývojári, ktorí sa zapájajú do projektu, posielajú do emailovej konferencie záplaty obsahujúce zmeny, opravy chýb a nové kódy. Každá knižnica, ovládač, prípadne iná samostatná časť má určeného správcu, ktorý zodpovedá za správnosť a kvalitu danej oblasti. Záplaty sú v rámci komunity diskutované. V prípade potreby sú upravované a posielané novšie verzie zahŕňajúce navrhnuté úpravy. Keď je záplata schválená, hlavný správca DPDK ju pridá do repozitára. Do konca roku 2015 boli vydávané nové verzie DPDK každé 4 mesiace. Od roku 2016 sa cyklus vydávania nových verzií skracuje na 3 mesiace. Prvý mesiac cyklu je vyhradený na posielanie nových zmien. Ďalší mesiac prebieha schvalovanie a začleňovanie zmien do hlavného repozitára. Posledný mesiac cyklu je vyhradený na testovanie a opravy chýb.

2.6.2 Adresárová štruktúra

V tomto oddieli je popísaná adresárová štruktúra aplikačného rámca DPDK verzie 16.04 a základné informácie o dôležitých adresároch. DPDK obsahuje nasledujúce adresáre:

- *app* – zdrojové kódy aplikácií, ktoré slúžia na testovanie DPDK alebo ovládačov

[12] *Non-uniform Memory Acces*
[13] GNU General Public License
[14] *Environment Abstraction Layer*

- *config* – šablóny pre tvorbu konfiguračných súborov, ktoré umožňujú aktivovať/deaktivovať rôzne položky pre knižnice ako napríklad voľby pre ladenie
- *doc* – zdrojové texty vo formáte RST[15] pre generovanie dokumentácie
- *drivers* – zdrojové kódy ovládačov hardvérových alebo virtuálnych zariadení, v adresári *drivers/net* sa nachádzajú sieťové zariadenia, v adresári *drivers/crypto* kryptografické zariadenia
- *examples* – zdrojové kódy aplikácií, ktoré znázorňujú príklady použitia knižníc
- *lib* – zdrojové kódy hlavných knižníc
- *mk* – súbory typu *makefile* dôležité pre prekladový systém
- *pkg* – súbory pre tvorbu balíčkov
- *scripts* – rôzne skripty uľahčujúce vývoj knižníc
- *tools* – užívateľské nástroje

2.6.3 EAL

Vrstva EAL riadi prístup k nízkoúrovňovým prostriedkom. Poskytuje ostatným knižniciam a aplikáciám postaveným na DPDK rozhranie zapúzdrujúce špecifiká prostredia pre prístup k hardvérovým zdrojom a pamäti. Medzi hlavné úlohy vrstvy EAL patrí:

- spustenie, načítanie a inicializácia DPDK
- riadenie rozdeľovania úloh na jadrá procesoru
- správa pamäte
- rozhranie pre prístup k PCI prostriedkom
- ladiace a zaznamenávacie funkcie
- určenie potrebných vlastností procesoru
- obsluha prerušení a budíkov

Po spustení DPDK aplikácie sa pred zavolaním funkcie `main()` skontroluje, či je typ architektúry zvolený konfiguračným súborom podporovaný procesorom. Následne sa vo funkcii `main()` musí volať funkcia `rte_eal_init()`, ktorá má na starosti inicializáciu a spustenie vrstvy EAL spoločne s vytvorením a inicializáciou vlákien. Argumenty príkazového riadku sa rozdeľujú na EAL argumenty, ktoré sú spoločné pre všetky aplikácie využívajúce vrstvu EAL, a aplikačne špecifické argumenty, ktoré sú od EAL argumentov oddelené pomocou „--“. Pre vytváranie EAL vlákien[16] sa používa knižnica *pthread*. Každému EAL vláknu je pridelené jednoznačné číslo, ktoré je uložené v pamäti lokálnej pre vlákno. EAL vlákno je *pthread* vlákno s unikátnym identifikátorom. Zvyčajne je pridelené jedno vlákno na jedno

[15] *reStructuredText*
[16] v DPDK označované aj ako *lcore*

jadro procesoru, aby sa ušetril čas pri prepínaní kontextu. Pri použití väčšieho počtu vlákien, môže byť na jedno fyzické jadro procesoru naviazaných viac vlákien. Pridelenie vlákien konkrétnym fyzickým jadrám je možné pomocou EAL argumentu príkazového riadku *--lcores*.

DPDK používa na prideľovanie blokov fyzickej pamäte súborový systém *hugetlbfs*[17]. EAL riadi prideľovanie a mapovanie pamäte z *hugetlbfs*. Rezervovanie súvislých blokov fyzickej pamäte je zabezpečené pomocou pamäťových zón (*rte_memzone* API). Je možné špecifikovať zarovnanie počiatočnej adresy blokov pamäte. EAL poskytuje aj rozhranie `rte_malloc()` funkcií pre prideľovanie pamäte z veľkých stránok. Funkcie z rodiny `rte_malloc()` by sa však mali používať iba pri konfiguračných kódoch a nie pri spracovaní dát, pretože sú pomalé a používajú zámky. Pri spracovaní dát je odporúčané používať API, ktoré poskytuje knižnica *librte_mempool* [14].

2.6.4 Knižnica mempool

Mempool je prideľovač pamäte, ktorý pracuje s objektami pevnej veľkosti. Identifikovaný je podľa mena. Pre ukladanie objektov používa knižnicu *librte_ring*, ktorá umožňuje spravovanie front typu *rte_ring*. Zvýšenie výkonnosti dosahuje *mempool* použitím vyrovnávacej pamäti pre jadrá a zarovnaním objektov v pamäti tak, aby boli vhodne rozložené s ohľadom na pamäťové kanály. *Mempool* nájde uplatnenie tam, kde sa vyžaduje vysoká výkonnosť a je potreba používať objekty pevnej veľkosti. V rámci DPDK knižníc sa *mempool* používa v knižnici *librte_mbuf* a vo vrstve EAL pre logovacie služby. Pomocou konfiguračného nastavenia `CONFIG_RTE_LIBRTE_MEMPOOL_DEBUG` je možné zapnúť ladiaci mód. Jednou zo služieb, ktoré poskytuje ladiaci mód, je zber štatistík o počte objektov prideľovaných z *mempoolu* a ukladaných do *mempoolu*. Okrem toho je v ladiacom móde pridané na začiatok a koniec každého bloku *cookie*[18], ktoré dodáva ochranu prepisovania a umožňuje ladenie pretečenia vyrovnávacích pamätí a dvojitého uvoľňovania [14].

2.6.5 Knižnica mbuf

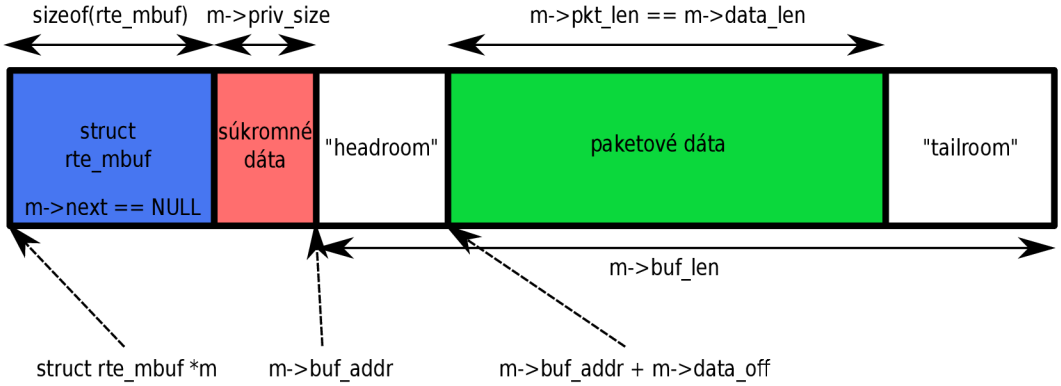
Knižnica *librte_mbuf* riadi alokovanie a uvoľňovanie bufferov pre správy. Buffer je reprezentovaný štruktúrou `struct rte_mbuf` (pre zjednodušenie je v ďalšom texte označovaná ako *mbuf*). Názov týchto bufferov je odvodený z anglického spojenia *Message Buffer*. Knižnica *librte_mbuf* používa prideľovač pamäte *mempool*. *Mbuf* obsahuje položku určujúcu *mempool*, z ktorého pochádza. Podľa toho sa pri uvoľňovaní vráti do správneho *mempoolu*. Štruktúra `rte_mbuf` môže obsahovať sieťový paket alebo kontrolný buffer. Príznak `CTRL_MBUF_FLAG` určuje, či sa jedná o kontrolný *mbuf* alebo nie. Konfiguračná konštanta `CONFIG_RTE_MBUF_DEBUG` umožňuje zapnúť ladiaci režim, v ktorom sa pred každou operáciou s *mbufom* vykonávajú rôzne kontroly správnosti [14].

Štruktúra bufferu Štruktúra `rte_mbuf` zaberá dva riadky vyrovnávacej pamäte, pričom najčastejšie používané položky sa nachádzajú v prvom riadku. Za metadátami umiestnenými v štruktúre `rte_mbuf` sa nachádza oblasť pevnej veľkosti určená pre paketové dáta. Výhodou tejto pamäťovej reprezentácie bufferu pre paketové dáta je, že stačí iba jedna

[17] súborový systém poskytujúci rozhranie pre používanie veľkých stránok (*huge pages*) – stránok väčších ako 4 kB

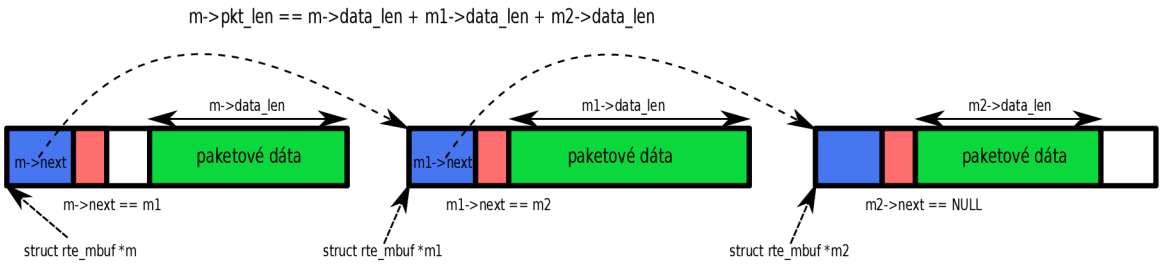
[18] číslo typu `uint64_t` s konkrétne určenou hodnotou

operácia pre alokovanie, resp. uvoľnenie celého bufferu vrátane metadát a samotných paketových dát. Metadáta obsahujú kontrolné informácie, medzi ktoré patrí napríklad typ, dĺžka dát uložených v danom *mbufe*, celková dĺžka dát uložených vo všetkých zreťazených *mbufoch*, vzdialenosť začiatku paketových dát od začiatku bufferu a ukazovateľ umožňujúci zreťaziť niekoľko *mbufov*. Na obrázku 2.2 je zobrazená štruktúra *mbufu*.



Obr. 2.2: Znázornenie štruktúry *mbufu*

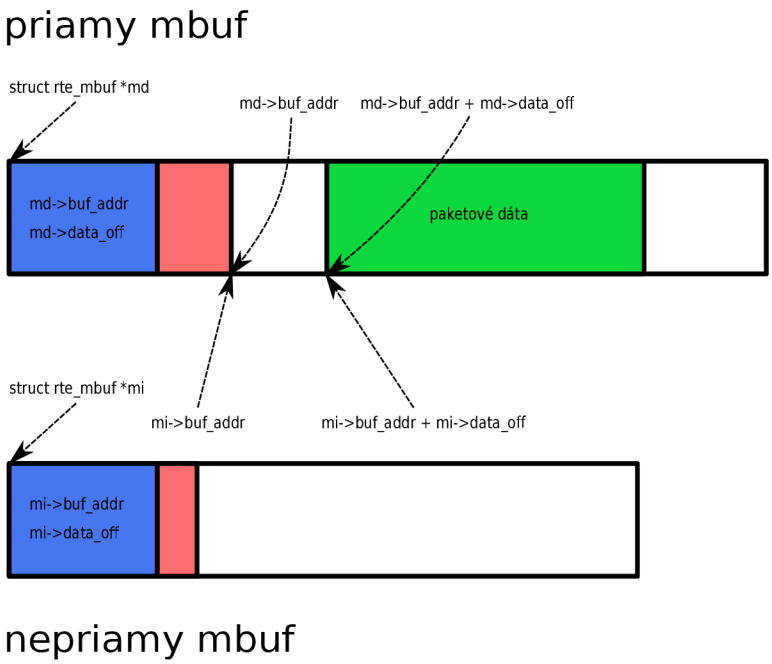
Zreťazenie *mbufov* sa používa pre uloženie paketov, ktorých dĺžka presahuje veľkosť bufferu pre uloženie paketových dát. Toto sa využíva napríklad pre *jumbo* rámce. Príklad zreťazených *mbufov* je na obrázku 2.3. Pri uvoľňovaní *mbufu*, ktorý obsahuje niekoľko zreťazených segmentov, sú uvoľnené všetky. Dáta v bufferi nezačínajú hneď na začiatku oblasti určenej pre uloženie paketových dát, ale od začiatku bufferu sa necháva voľné miesto označované *headroom*. Toto voľné miesto sa môže využiť napríklad pri pripájaní protokolových hlavičiek pred začiatok paketových dát.



Obr. 2.3: Znázornenie zreťazených *mbufov*

Priame a nepriame buffere Buffere môžu byť priame alebo nepriame. Priamy *mbuf* je kompletne nezávislý, ukazovateľ na buffer a vzdialenosť začiatku paketových dát má nastavenú podľa svojich vlastných dát. Naproti tomu nepriamy *mbuf* má ukazovateľ na buffer a vzdialenosť začiatku paketových dát nastavenú podľa iného priameho *mbufu*. Rozdiel medzi priamym a nepriamym *mbufom* je zobrazený na obrázku 2.4. Toto sa dá využiť napríklad pri potrebe duplikovať pakety. Každý *mbuf* obsahuje referenčný čítač. *Mbuf* sa stáva nepriamym, keď sa pripojí k priamemu. Pri pripojení *mbufu* k priamemu sa u priameho *mbufu* zvýši referenčný čítač o 1. Pri odpojení sa referenčný čítač o 1 zmenší. Keď má referenčný čítač *mbufu* hodnotu 0, *mbuf* sa uvoľní. Nie je možné pripojiť nepriamy *mbuf* na iný nepriamy. Taktiež nie je možné, aby sa stal *mbuf* nepriamym, keď je hodnota jeho

referenčného čítača väčšia ako 1, čo znamená, že je naň pripojený iný nepriamy *mbuf*. Pripojiť *mbuf* sa dá pomocou funkcie `rte_pktmbuf_attach()` alebo `rte_pktmbuf_clone()`, ktorá dokáže naklonovať aj buffere s väčším počtom segmentov. Pre odpojenie slúži funkcia `rte_pktmbuf_detach()`.



Obr. 2.4: Znázornenie rozdielu medzi priamym a nepriamym *mbufom*

2.6.6 Ovládač sieťového zariadenia

DPDK používa pre prístup k sieťovým zariadeniam ovládače bežiace v užívateľskom priestore používajúce *polling*. Ovládač tvorí API pre ethernetové zariadenie[19], ktoré poskytuje knižnica *librte_ether*. Ovládač musí implementovať sadu funkcií pre príjem, odosielanie paketov, inicializáciu a konfiguráciu zariadenia. Okrem toho je možné podporovať široké rozpätie funkcií pre filtrovanie, zobrazovanie štatistík a informácií o zariadení. Funkcie sú pomocou funkčných ukazovateľov sprístupnené DPDK aplikácii pomocou jednotného rozhrania.

Každé sieťové zariadenie je reprezentované štruktúrou `struct rte_eth_dev`. Na začiatku tejto štruktúry sa nachádzajú ukazovatele na funkcie pre príjem a odosielanie dát a ukazovateľ na štruktúru `struct rte_eth_dev_data`, v ktorej sú uložené dáta zariadenia. Ukazovatele na ďalšie funkcie implementované ovládačom sú zoskupené v štruktúre `struct eth_dev_ops`. Ukazovateľ na túto štruktúru sa taktiež nachádza v štruktúre `struct rte_eth_dev`.

Modely spracovania paketov DPDK podporuje nasledujúce dva modely spracovania paketov:

- **„*Run-to-completion*“ model**

  Jedno logické jadro zabezpečuje príjem, spracovanie a odoslanie paketov z jednej

[19] *Ethernet Device API*

fronty na jednom porte. Pakety sú daným jadrom najskôr prijaté pomocou API pre príjem paketov, následne sú spracované a potom odoslané pomocou API pre odosielanie paketov. Niekoľko jadier môže takto obsluhovať niekoľko rôznych front na rôznych portoch.

- **Model zreťazeného spracovania**

  Jedno logické jadro prijíma pakety z jednej alebo viacerých front. Prijaté pakety posúva ďalšiemu jadru na spracovanie pomocou API štruktúry *ring*. Takto môžu byť pakety presúvané medzi viacerými jadrami, až kým sa nedostanú k jadru, ktoré pomocou API pre odosielanie paketov zabezpečí ich odoslanie.

Princípy štruktúry ovládača Ovládače používajú funkcie pre príjem a odosielanie zhlukov paketov. To umožňuje rozložiť réžiu spôsobenú volaním funkcie medzi väčšie množstvo paketov a používať špeciálne vlastnosti zariadení pre odosielanie väčšieho množstva dát v zhlukoch. Na druhú stranu prenášanie paketov v zhlukoch spôsobuje zvýšenie latencie. Ak je potrebné znížiť latenciu a je možné uspokojiť sa s nižšou priepustnosťou, dá sa zmenšiť veľkosť zhluku prijímaných a odosielaných paketov.

Funkcie pre príjem a vysielanie paketov implementované ovládačom sieťového zariadenia nepoužívajú zámky. Tieto funkcie nemôžu bežať paralelne na rôznych logických jadrách nad rovnakou frontou z určitého portu, avšak nad rôznymi frontami môžu bežať paralelne na viacerých jadrách. Tieto pravidlá musí zaistiť aplikácia používajúca API pre ethernetové zariadenia. Inicializačné a konfiguračné funkcie by mali byť vykonávané hlavným jadrom.

Reprezentácia paketov Pakety v DPDK sú reprezentované štruktúrou `rte_mbuf`. Sieťové zariadenia, ktoré umožňujú hardvérové spracovanie určitých časovo kritických operácií ako napríklad počítanie kontrolných súčtov IP hlavičiek, môžu ukladať do štruktúry `rte_mbuf` potrebné metadáta a poskytovať dané funkcie pomocou príslušných funkcií z API pre ethernetové zariadenia. Pri prijímaní paketov vypĺňa prijímacia funkcia položky *mbufu*. Pri odosielaní paketov sú pomocou položiek z *mbufu* vyplnené odosielacie deskriptory.

Typy sieťových ovládačov DPDK rozlišuje dva typy sieťových ovládačov:

- ovládač pre fyzické zariadenie (typ `PMD_PDEV`)
- ovládač pre virtuálne zariadenie (typ `PMD_VDEV`)

Pri inicializácii DPDK aplikácie prebieha skenovanie PCI zariadení podľa PCI identifikátorov. Keď existuje pre zariadenie ovládač, tak prebehne inicializácia daného zariadenia a je mu pridelené číslo a meno portu. Číslo portu slúži pre určenie zariadenia pri funkciách z API pre ethernetové zariadenia. Meno slúži pre ladiace a informatívne účely.

Virtuálne zariadenia sú inicializované na inom princípe. Pri spustení DPDK aplikácie je možné pridať virtuálne zariadenie pomocou argumentu príkazového riadku pre vrstvu EAL `--vdev`. Potom sa pri inicializácii aplikácie spustí inicializačná funkcia pre virtuálne zariadenie, ktorému je tiež priradené číslo a meno portu.

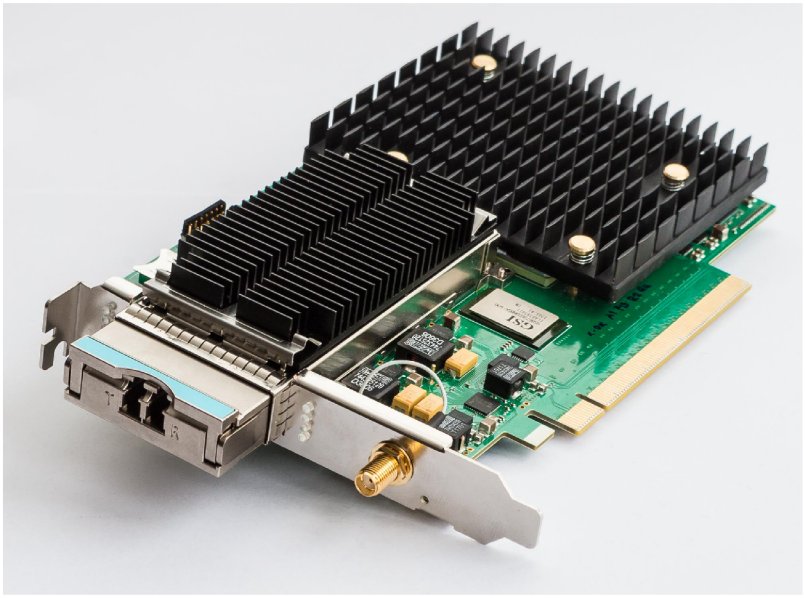
2.7 Rozhranie SZE2

SZE2 je rozhranie pre rýchly DMA prenos dát medzi sieťovou kartou a operačnou pamäťou hostiteľskej stanice. Rozhranie SZE2 bolo vyvinuté v rámci projektu *Liberouter* pre umož-

nenie rýchleho DMA prenosu dát cez zbernicu PCI-Express medzi sieťovými kartami rodiny COMBO a pamäťou RAM. Rozhranie je súčasťou hardvérovej akceleračnej platformy *NetCOPE*. Tvorbou aplikácií a rýchlymi dátovými prenosmi pre túto platformu sa zaoberali diplomové práce od Andreja Hanka [6] a Jiřího Slabého [10]. Na úrovni operačného systému sú použité moduly jadra ako ovládače sieťovej karty. Architektúra rozhrania je popísaná v oddiely 2.7.4. Rodina sieťových kariet COMBO je popísaná v oddiely 2.7.2. Pre tvorbu užívateľských aplikácií nad rozhraním SZE2 vznikla knižnica *libsze2* sprostredkúvajúca príjem a odosielanie dát. Knižnica *libsze2* je popísaná v oddiely 2.7.5.

2.7.1 Projekt Liberouter

Projekt *Liberouter* vznikol ako výskumná aktivita združenia CESNET v roku 2003. Pôvodným cieľom bola tvorba hardvérovo akcelerovaného IPv6 smerovača. V súčasnosti sa projekt zameriava na vývoj sieťových kariet pre Ethernet o rýchlostiach 10 Gb/s, 40 Gb/s, 100 Gb/s a hardvérovú akceleráciu nástrojov pre sieťovú bezpečnosť a monitorovanie. Sieťové karty a hardvérovo akcelerované nástroje sú založené na technológii FPGA. Okrem CESNETu sa na projekte podieľa Vysoké učení technické v Brne, Masarykova univerzita a České vysoké učení technické v Prahe. Vyvinuté nástroje sa nasadzujú v sieti CESNETu a stali sa základom pre komerčné produkty partnerských spoločností Netcope Technologies a Flowmon Networks [20].



Obr. 2.5: Sieťová karta COMBO-100G. Zdroj: galéria kariet projektu Liberouter [19].

2.7.2 Sieťové karty rodiny COMBO

Karty rodiny COMBO sú sieťové karty založené na technológii čipov FPGA. Prvé generácie kariet používali zbernice PCI a PCI-X, neskôr zbernicu PCI-Express. Súčasné karty generácie COMBOv3 používajú pre prenos dát do hostiteľského systému zbernicu PCI-Express. Medzi tieto karty patrí COMBO-100G (okrázok 2.5) a COMBO-80G. V týchto kartách sú použité FPGA čipy Virtex-7 od spoločnosti Xilinx. COMBO-100G obsahuje čip Virtex-7 H580T a používa zbernicu PCI-Express 3.0 x16. COMBO-80G obsahuje čip Virtex-7

690T a používa zbernicu PCI-Express 3.0 x8. FPGA čip na karte umožňuje meniť funkcionalitu karty načítaním novej konfigurácie do FPGA. Za účelom zjednodušenia vývoja akcelerovaných nástrojov s využitím kariet COMBO bola vytvorená platforma *NetCOPE*, ktorá poskytuje funkčné bloky pre DMA prenosy, sieťové rozhrania a ďalšie komponenty potrebné pre obsluhu karty. Pri tvorbe hardvérovo akcelerovanej aplikácie je tak potrebné doplniť len akceleračné jadro implementujúce časovo kritické operácie.

2.7.3 Nástroje pre prácu s kartou a firmvérom

Pre sieťové karty COMBO existuje sada softvérových nástrojov, ktoré uľahčujú vývoj, umožňujú konfiguráciu a využitie funkcií kariet. V tejto časti sú popísané nástroje, ktoré som použil pri testovaní a meraniach výkonnosti DPDK ovládača.

`csboot` Nástroj, ktorý umožňuje nahrať firmvér do karty.

`ibufctl` Nástroj pre prístup k registrom komponentu IBUF – vstupná sieťová vyrovnávacia pamäť. Tento nástroj umožňuje konfiguráciu, zobrazovanie a nulovanie štatistík z komponentu IBUF.

`obufctl` Nástroj pre prístup k registrom komponentu OBUF – výstupná sieťová vyrovnávacia pamäť. Umožňuje konfigurovať, zobrazovať a nulovať štatistiky z komponentu OBUF.

`hanicctl` Nástroj pre inicializáciu a konfiguráciu firmvéru HANIC.

`filterctl` Nástroj pre konfiguráciu filtra, ktorý je súčasťou firmvéru HANIC.

2.7.4 Architektúra rozhrania SZE2

Z pohľadu operačného systému je komunikácia so sieťovou kartou zabezpečená pomocou systémových ovládačov v podobe modulov jadra. Moduly jadra plnia dve základné funkcie:

- konfiguráciu, inicializáciu a ovládanie karty
- rýchle DMA prenosy medzi kartou a operačnou pamäťou

V rámci tejto práce je rozoberaný princíp paketových prenosov. Preto je ďalej bližšie preskúmaná druhá z uvedených úloh jadrových modulov.

Pre zjednodušenie tvorby aplikácií v užívateľskom priestore boli vytvorené knižnice, ktoré jednoduchým spôsobom sprístupňujú rozhranie poskytované modulmi jadra. Knižnici *libsze2*, ktorá sprístupňuje rozhranie SZE2 pre rýchle dátové prenosy, sa venuje oddiel 2.7.5. Okrem tejto knižnice existujú ešte knižnice *libcommlbr*, *libcombo* a *libpcap-sze2*. Knižnica *libcommlbr* obsahuje základné, všeobecné funkcie pre ladenie a operácie, ktoré sa často používajú v aplikáciách postavených na platforme *NetCOPE*. Knižnica *libcombo* obsahuje funkcie pre bootovanie firmvéru, inicializáciu a konfiguráciu karty. Knižnica *libpcap-sze2* rozširuje knižnicu *libpcap* o podporu rozhrania SZE2.

Ovládače Štruktúra ovládačov platformy *NetCOPE* je tvorená viacerými vrstvami. Vrstvy zabraňujú potrebe duplikovať zdrojové kódy a zapúzdrujú špecifiká jednotlivých kariet. V súčasnosti sa pre karty generácie COMBOv3 používajú moduly:

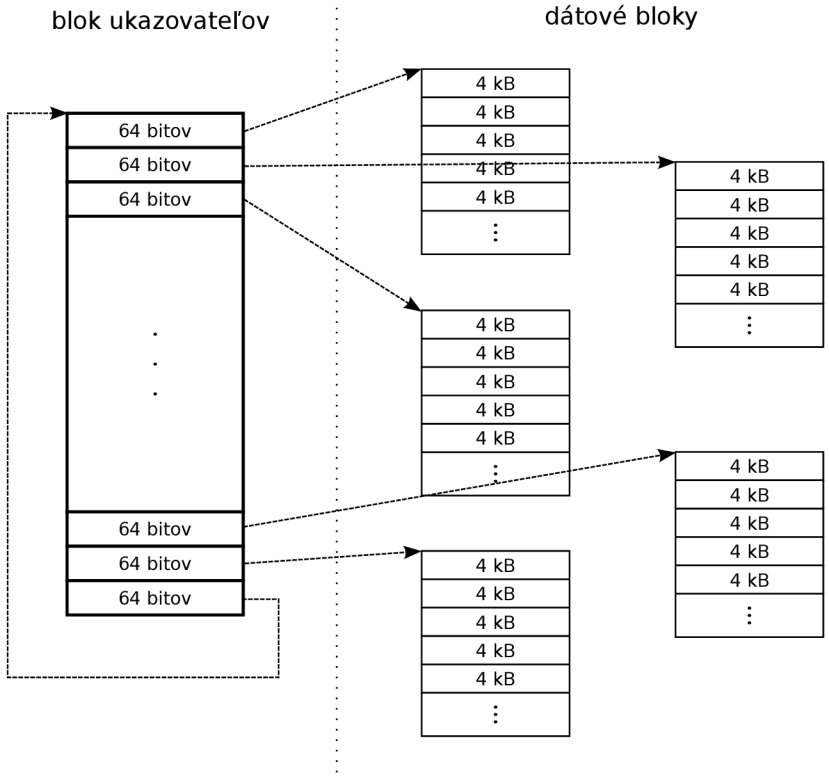
- *combov3* – tvorí najnižšiu vrstvu, ktorá implementuje operácie špecifické pre konkrétnu skupinu kariet, v tomto prípade generácie COMBOv3
- *combo6core* – tvorí ďalšiu vrstvu, ktorá spravuje zoznamy pripojených kariet a aplikačných ovládačov, prepojuje dolnú vrstvu s aplikačnými modulmi
- *szedata2*
- *szedata2_cv3*

Moduly *szedata2* a *szedata2_cv3* predstavujú aplikačný ovládač. Modul *szedata2_cv3* tvorí časť ovládača, ktorá závisí na použitom hardvéri. Prostredníctvom rozhrania modulu *combo6core* sprostredkúva komunikáciu modulu *szedata2* s hardvérom sieťovej karty. Modul *szedata2* tvorí časť ovládača, ktorá je na hardvéri nezávislá. Je ovládačom znakového zariadenia reprezentovaného súborom */dev/szedataIIX*, kde *X* je číslo zariadenia. Pre užívateľský priestor zapuzdruje pomocou štruktúry obsahujúcej ukazovatele na obslužné rutiny hardvérovo špecifického ovládača rozhranie pre komunikáciu s hardvérom.

Užívateľské aplikácie komunikujú s modulom *szedata2* prostredníctvom systémových volaní. Ovládač neimplementuje volania `read()` a `write()`. Prenos dát je zabezpečený použitím volania `mmap()`, ktorým je pamäť kruhového bufferu namapovaná do užívateľského priestoru aplikácie, a špecifickými operáciami *szedata2* zariadenia, ktoré sú sprístupnené volaním `ioctl()`. Medzi tieto operácie patrí hlavne špecifikácia kruhových bufferov, s ktorými bude aplikácia pracovať, zapnutie a vypnutie prenosov, zamknutie a odomknutie bufferu pre čítanie a zápis dát.

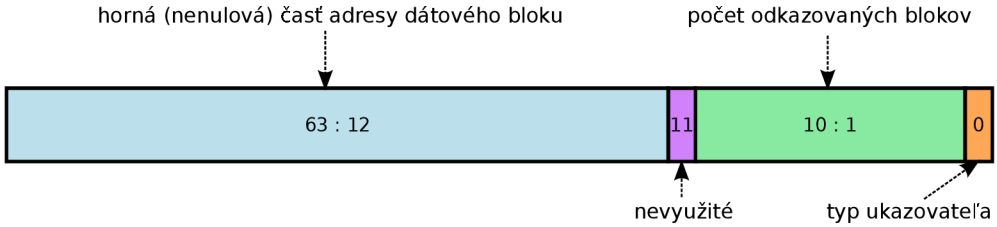
Princíp DMA prenosov Na úrovni firmvéru je prenos dát medzi kartou a operačnou pamäťou zabezpečený pomocou DMA radiča implementovaného v FPGA a dvojice kruhových bufferov, pričom jeden z dvojice sa nachádza v operačnej pamäti RAM hostiteľského systému a druhý v pamäti na karte. Prijímací aj vysielací smer sú oddelené komunikačné kanály, ktoré obsahujú svoju vlastnú dvojicu kruhových bufferov. V každom smere môže byť jeden alebo viac kanálov s vlastnými bufferami. Počet kanálov závisí od konkrétneho typu firmvéru. Obsadenosť kruhového bufferu je sledovaná a riadená pomocou štvorice ukazovateľov – dvojica ukazovateľov na začiatok a koniec dát pre buffer v RAM a druhá dvojica pre buffer na karte. DMA radič iniciuje, riadi prenosy dát a komunikuje tak s ovládačom *szedata2* na základe hodnôt týchto štyroch ukazovateľov.

Kruhový buffer Celková veľkosť kruhového bufferu v pamäti RAM dosahuje rádovo desiatky až stovky MB. V diplomovej práci Andreja Hanka [6] je odvodená veľkosť kruhového bufferu. Kruhový buffer sa skladá z tisíciek blokov pamäte o veľkosti 4 kB, čo zodpovedá veľkosti stránky na architektúrach x86. Tieto bloky musia spĺňať podmienky pre DMA prenosy. Adresy blokov sú zarovnané na hranice 4 kB, čiže spodných 12 bitov adresy bloku je pri adresovaní bezvýznamných. Tieto spodné bity boli využité pri optimalizácii štruktúry bufferu pre dosiahnutie prenosov o rýchlosti 100 Gb/s. V tomto sa súčasná štruktúra bufferu líši od štruktúry popisovanej v diplomovej práci Andreja Hanka. Pre popis bufferu sú použité ukazovatele na bloky dát, ktoré sú usporiadané do lineárneho zoznamu a uložené v bloku obsahujúcom ukazovatele, ktorý musí byť tiež schopný DMA



Obr. 2.6: Štruktúra kruhového bufferu

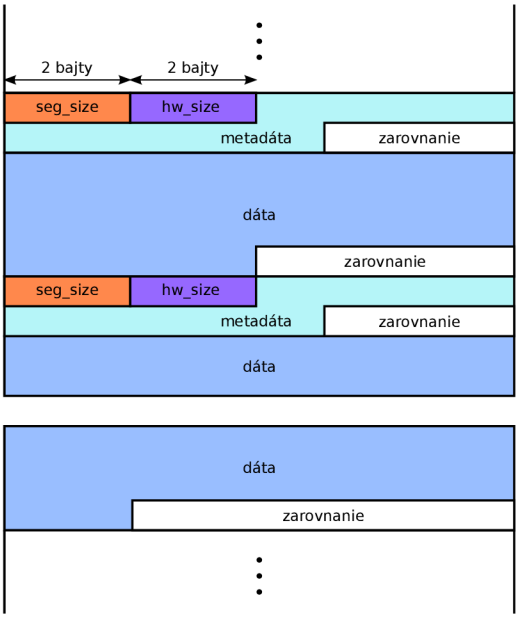
prenosov. Pri alokovaní dátových blokov sa alokujú súvislé oblasti obsahujúce viac ako len jeden blok. Bity ukazovateľa 1 – 11 sú potom použité na označenie počtu blokov nachádzajúcich sa za odkazovaným dátovým blokom. Bit 0 je použitý na rozlíšenie typu ukazovateľa. Posledný ukazovateľ v bloku neukazuje na dátový blok ale na začiatok bloku ukazovateľov. Na obrázku 2.6 je znázornená štruktúra kruhového bufferu pomocou dátových blokov a ukazovateľov. Obrázok 2.7 ilustruje formát ukazovateľa.



Obr. 2.7: Formát ukazovateľa

Vytvorenie kruhového bufferu vrátane blokov s ukazovateľmi v pamäti RAM zaisťuje ovládač. Po vytvorení bufferu informuje DMA radič o veľkosti bufferu a adrese na začiatok bloku obsahujúceho ukazovatele. Po ukončení prenosov je úlohou ovládača uvoľniť pamäť kruhového bufferu.

Formát dát Rozhranie SZE2 umožňuje prenos segmentov dát rôznej dĺžky. Kedže sú segmenty uložené v pamäti súvisle za sebou, je možné krátke segmenty prenášať agregovane aj niekoľko naraz. Segment dát je tvorený hlavičkou a samotnými dátami. Dáta potom



Obr. 2.8: Formát SZE2 segmentu s detailom uloženia na hranici dátových blokov

obsahujú typicky rámec zo spojovej vrstvy. Hlavička obsahuje položky *seg_size*, *hw_size* a metadáta. Položka *seg_size* má veľkosť 2 B a označuje dĺžku celého segmentu vrátane zarovnanej hlavičky. Položka *hw_size* má veľkosť 2 B a označuje dĺžku metadát. Metadáta môžu obsahovať informácie z akceleračného jadra. Hlavička aj dáta sú zarovnané na 8 B. Kedže segmenty dát majú premenlivú dĺžku, poloha nasledujúceho segmentu je vždy známa až po spracovaní hlavičky aktuálneho segmentu. Na obrázku 2.8 je zobrazený formát segmentu a rozloženie segmentov na hranici blokov pamäte.

2.7.5 Knižnica libsze2

Knižnica *libsze2* zapúzdruje rozhranie modulu jadra *szedata2* a vytvára jednoduché rozhranie pre tvorbu užívateľských aplikácií. Knižnica sa skladá z troch hlavných častí – modulov:

- *common* – obsahuje dôležité riadiace funkcie pre inicializáciu, ladiace funkcie a nízkoúrovňové funkcie obalujúce systémové volania
- *read* – funkcie pre čítanie dát
- *write* – funkcie pre zápis dát

Inicializácia a ďalšie riadiace funkcie Inicializácia a spustenie prenosov prebieha typicky v nasledujúcich krokoch:

1. Zavolanie funkcie:

   ```
   struct szedata *szedata_open(const char *node);
   ```

   Funkcii sa ako parameter zadá cesta k znakovému zariadeniu *szedata2*, nad ktorým budú prebiehať prenosy (napríklad */dev/szedataII0*). Funkcia vráti ukazovateľ na štruktúru `struct szedata`, ktorá sa následne predáva ako parameter všetkým ostatným volaniam z knižnice.

2. Výber prijímacích a vysielacích DMA kanálov volaním funkcie:

```
int szedata_subscribe3(struct szedata *sze, __u32 *rx,
   __u32 *tx);
```

   Funkcii sa pomocou parametrov *rx* (prijímací smer) a *tx* (vysielací smer) zadáva maska DMA kanálov, ktoré sa majú použiť. Funkcia sa pokúsi kanály pripraviť a parametre nastaví na hodnotu masky kanálov, ktoré sa jej skutočne podarilo pripraviť.

3. Naštartovanie prenosov volaním funkcie:

```
int szedata_start(struct szedata *sze);
```

Ukončenie prenosov a uvoľnenie zdrojov zabezpečuje funkcia `szedata_close()`.

Funkcia `szedata_ifaces_available()` umožňuje zistiť počet dostupných prijímacích a vysielacích kanálov.

Prijímanie dát Prijímanie dát sprostredkúvajú funkcie:

- `szedata_read_next()`
- `szedata_read_next_noblock()`

Tieto funkcie vrátia ukazovateľ na dáta jedného segmentu. Segment je potrebné spracovať pred nasledujúcim volaním funkcie. V prípade, keď v kruhovom bufferi nie sú dostupné žiadne nové dáta, funkcia `szedata_read_next_noblock()` hneď vráti `NULL`. Blokujúca funkcia `szedata_read_next()` v takom prípade volá volanie `poll()` zaobalené vo funkcii `szedata_poll()` a až v prípade neúspechu vráti `NULL`.

Prijímanie dát je možné aj s použitím nízkoúrovňových funkcií `szedata_rx_lock_data()` a `szedata_rx_unlock_data()`, ktoré umožňujú zamknúť a odomknút časť kruhového bufferu pre čítanie dát. Tento spôsob je použitý pri implementácii DPDK ovládača, aby bolo možné prijať väčšie množstvo segmentov dát naraz a znížiť réžiu volania funkcie `szedata_read_next()`.

Odosielanie dát Pre odosielanie dát je možné použiť funkcie:

- `szedata_prepare_and_try_write_next()`
- `szedata_burst_write_next()`
- `szedata_prepare_packet()`
- `szedata_try_write_next()`

Odosielať dáta je tiež možné aj prostredníctvom nízkoúrovňových funkcií pre zamykanie a odomykanie kruhového bufferu, `szedata_tx_lock_data()` a `szedata_tx_unlock_data()`. Rovnako ako varianty pre prijímanie dát sú tieto funkcie použité pri implementácii DPDK ovládača, aby bolo možné znížiť réžiu volania funkcií.

Kapitola 3

Návrh a princíp podpory kariet COMBO v DPDK

Pridanie podpory nového sieťového zariadenia do DPDK je možné pomocou vytvorenia nového ovládača. Táto kapitola popisuje návrh ovládača do DPDK pre sieťové karty COMBOv3. V sekcii 3.1 sú porovnané vlastnosti DPDK a SZE2. Sekcia 3.2 popisuje návrh architektúry DPDK ovládača *szedata2*.

Ovládač je postavený nad knižnicou *libsze2*. Názov ovládača bol zvolený podľa príslušného modulu jadra, ktorý zabezpečuje prenosy rozhraním SZE2. V sekcii 3.3 sú ukázané výsledky meraní réžie DPDK pri prijímaní a odosielaní paketov. Namerané hodnoty predstavujú teoretické limity, ku ktorým sa ovládač *szedata2* snaží priblížiť.

3.1 Porovnanie princípov DPDK a SZE2

Táto sekcia poskytuje súhrn základných princípov a vlastností, v ktorých sa DPDK a SZE2 zhodujú, ale taktiež sú zdôraznené ich rozdiely. Zameral som sa na spôsob použitia princípu nulového kopírovania, ktorý je spoločný pre obidva systémy, ale spôsob aplikácie tejto techniky sa líši. Ďalej je porovnané a priblížené rozhranie pre prenosy zhlukov paketov, použitie viacerých vstupno-výstupných front a spôsob správy bufferov pre ukladanie paketov.

Nulové kopírovanie DPDK používa pre ukladanie paketov *mbufy* (popísané v sekcii 2.6.5). Veľkosť *mbufov* je pevne daná a je určená počas inicializácie pri ich vytvorení. Sieťové zariadenia podporované v DPDK používajú pre DMA prenosy deskriptory. V RX smere ovládač nastaví do deskriptoru fyzickú adresu pamäťového miesta pre paket vo voľnom *mbufe*, počká kým sa pomocou DMA prenosov paket do *mbufu* prenesie a sprístupní paket aplikácii. V TX smere ovládač nastaví do deskriptoru fyzickú adresu, na ktorej sa nachádza paket v *mbufe*, a dĺžku paketu. Opäť počká, kým je paket cez DMA prenosy presunutý do sieťovej karty, a uvoľní *mbuf* pre ďalšie použitie. *Mbufy* obsahujú pred a za samotnými dátami paketu voľné miesto, ktoré je možné využiť na pripájanie rôznych hlavičiek, kontrolných súčtov, prípadne iných informácií, ktoré si chce užívateľská aplikácia uložiť. Využitie tohto miesta je napríklad pri šifrovaní. Možnosť používať nepriame *mbufy* odstraňuje potrebu kopírovať pakety v prípadoch, kedy je potrebné paket duplikovať. Keďže pakety sa nachádzajú na rôznych adresách a ich DMA prenosy sú založené na princípe deskriptorov, každý paket je prenášaný samostatne. Samostatné prenosy krátkych paketov znižujú efektivitu využitia prenosového pásma zbernice.

Rozhranie SZE2 pracuje s veľkými súvislými pamäťovými blokmi. Pakety sa v týchto blokoch nachádzajú jeden za druhým. Voľné miesto vzniká len kvôli zarovnaniu SZE2 segmentov, ktorých formát je popísaný na obrázku 2.8 na strane 20. Vďaka súvislému uloženiu paketov je možné efektívnejšie využívať zbernicu aj pre krátke pakety, kedže je možné prenášať niekoľko paketov naraz. SZE2 buffere sú mapované do užívateľského priestoru. Potom, čo je paket prenesený cez DMA prenosy do SZE2 bufferu, aplikácia môže pristupovať priamo k paketovým dátam. Tento prístup je vhodný, keď sú pakety spracúvané jeden za druhým a po spracovaní je vždy možné ich uvoľniť. Problém nastáva v prípade, kedy by bolo nutné pozdržať uvoľnenie nejakého paketu. Kým nie je daný paket spracovaný, nie je možné posunúť ukazovateľ a uvoľniť tak miesto v bufferi. Ak sú za takýmto paketom už spracované pakety, nie je možné využiť ani ich pamäťové miesto.

Zhluky paketov Funkcie pre príjem a vysielanie paketov, ktoré sú súčasťou DPDK API sú postavené pre prácu so zhlukmi paketov. Parametrom funkcie sa zadáva počet paketov pre prijatie, resp. odoslanie. Veľkosť zhlukov si tak môže prispôsobiť aplikácia. Čím väčšie zhluky sú použité, tým je väčšia latencia. Je možné posielať pakety aj po jednom a minimalizovať tak latenciu.

API knižnice *libsze2* obsahuje funkcie pre príjem a odosielanie dát vyššej a nižšej úrovne. Funkcie sú popísané v oddieli zaoberajúcom sa knižnicou *libsze2* (2.7.5). Nízkoúrovňové funkcie pre zamykanie a odomykanie bufferov priamo volajú `ioctl()` volanie, ktoré má na starosti posúvanie ukazovateľov ohraničujúcich dáta v SZE2 bufferoch. V RX smere je pred prijatím dát použité volanie, ktoré zabezpečí, že modul jadra ohraničí oblasť, z ktorej je možné dáta prečítať. Aplikácia následne môže z tejto oblasti čítať pakety. Keď sú pakety vyčerpané, aplikácia musí zavolať volanie, ktorým je spracovaná oblasť uvoľnená, čo znamená, že sa posunie ukazovateľ a do danej oblasti je možné preniesť nové dáta z karty. V TX smere modul jadra ohraničí oblasť, do ktorej môže aplikácia zapisovať pakety. Aplikácia potom do tejto oblasti ukladá pakety jeden za druhým. Keď je vyčerpané voľné miesto v tejto oblasti, aplikácia musí zavolať volanie, ktoré zabezpečí presun dát z danej oblasti do karty a posunutie ukazovateľa. Po úspešnom presune dát je možné do danej oblasti znovu zapisovať nové dáta.

Vstupno-výstupné fronty DPDK umožňuje používať viaceré RX a TX fronty. Úlohou ovládača je poskytovať informácie o počte dostupných RX a TX front. Aplikácia si potom môže nastaviť, koľko front bude používať, a na akých jadrách budú jednotlivé fronty spracovávané. Funkcie pre príjem a odosielanie paketov sa vždy volajú nad konkrétnou frontou.

Pred spustením prenosov cez rozhranie SZE2 je potrebné zavolať funkciu, pomocou ktorej sa modulom jadra posunie informácia, ktoré DMA kanály chce daná aplikácia používať pre RX a TX smer. Takto je možné presne namapovať konkrétne DMA kanály rozhrania SZE2 na jednotlivé fronty v DPDK.

Správa bufferov DPDK aj rozhranie SZE2 používajú predalokovanie bufferov pred samotným spustením prenosov. DPDK používa vlastného správcu pamäťových objektov pevnej veľkosti – *mempool*. *Mempool* a *mbufy*, ktoré spravuje sú vytvorené pri spustení a inicializácii DPDK aplikácie. Pri prijímaní dát sú *mbufy* odoberané z *mempoolu* a pri odosielaní paketov zase vrátené späť.

Buffere, ktoré používa SZE2 sú alokované pri zavedení príslušného modulu jadra, prípadne pri prvom spustení aplikácie, ktorá ich má používať. Toto je možné nastaviť para-

metrom jadrového modulu. Voľné a obsadené miesto v bufferoch je ohraničené pomocou ukazovateľov na začiatok a koniec dát, ktoré sú spravované modulom jadra.

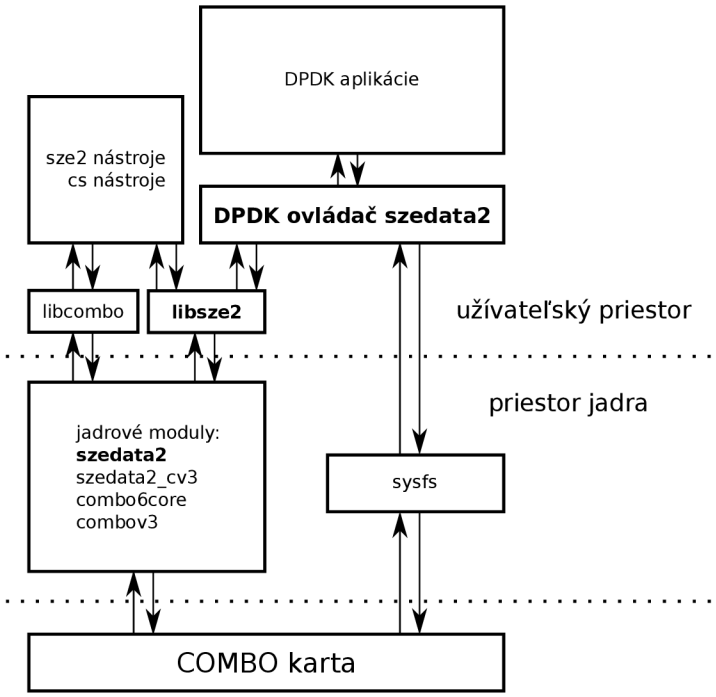
3.2 DPDK ovládač szedata2

Ovládač nad rozhraním SZE2 pre COMBOv3 karty je možné implementovať v dvoch variantoch:

- Ovládač závislý na knižnici *libsze2* a moduloch jadra používajúci API knižnice *libsze2*. Výhodou tohto ovládača je jednoduchšia implementácia. Nevýhodou sú externé závislosti, ktoré nie sú súčasťou DPDK.
- Ovládač implementujúci funkcionalitu modulov jadra priamo komunikujúci prostredníctvom mapovania adresného priestoru karty do užívateľského priestoru. Výhodou je odstránenie externých závislostí a možnosť efektívnejšie implementovať mechanizmy riadenia súbežného prístupu k SZE2 bufferom s ohľadom na bezzámkový princíp DPDK. Nevýhodou je náročnejšie implementácia, kedže je nutné implementovať od základov komunikáciu s kartou.

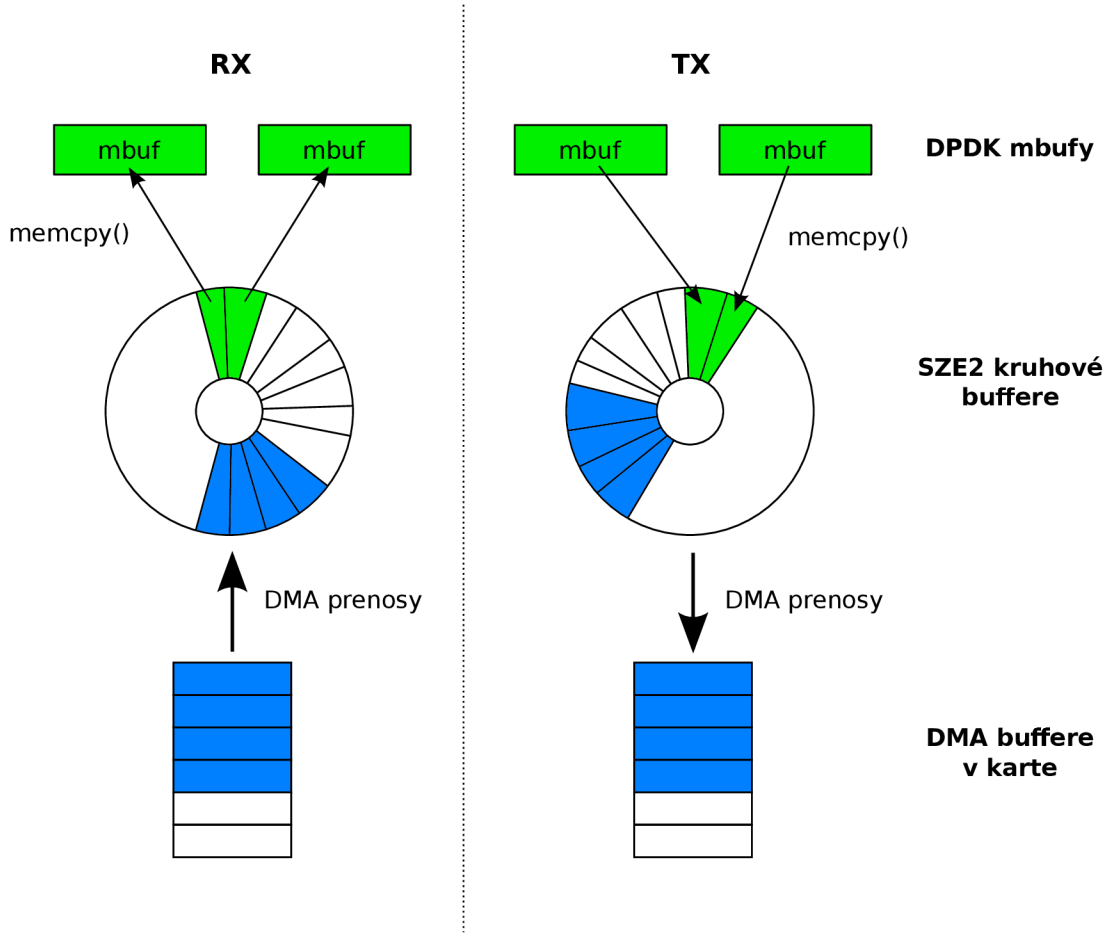
V rámci tejto práce bol implementovaný prvý z uvedených variantov. Druhý variant je možné implementovať ako pokračovanie práce. Vytvorený ovládač bol nazvaný *szedata2*, podľa názvu modulu jadra, ktorý zabezpečuje prenosy rozhraním SZE2. Popis implementácie ovládača je v kapitole 4.

Obrázok 3.1 znázorňuje hierarchickú štruktúru aplikácie využívajúcej DPDK API pre spracovanie paketov nad sieťovou kartou COMBO. Paralelne s DPDK aplikáciou môžu byť používané aj existujúce SZE2 nástroje.



Obr. 3.1: Hierarchická štruktúra vrstiev DPDK aplikácií postavených nad kartami COMBO.

Aplikácie v DPDK pracujú s paketmi uloženými v *mbufoch*. Ovládač *szedata2* musí zaistiť, aby boli v RX smere *mbufy* naplnené novými paketmi a v TX smere pakety z *mbufov* odoslané do sieťovej karty. Ako bolo spomenuté v sekcii 3.1, pakety prijaté rozhraním SZE2 sú umiestnené v SZE2 bufferi. Aby bolo možné s týmito paketmi pracovať ľubovoľným spôsobom bez obmedzenia DPDK aplikácie, je potrebné pakety kopírovať zo SZE2 bufferu do *mbufov*. Manipulácia s ukazovateľom na paketové dáta v *mbufe* tak, aby ukazoval na dáta v SZE2 bufferi, nie je možná kvôli blokovaniu SZE2 bufferu v prípade pozdržaného spracovania niektorých paketov, ale aj kvôli udržaniu voľného miesta pred a za dátami paketu pre prípadné pripájanie dát. V TX smere je potrebné dáta kopírovať z *mbufu* do SZE2 bufferu, kedže rozhranie SZE2 neumožňuje odosielanie paketov pomocou deskriptorov. Pakety na odoslanie musia byť umiestnené v SZE2 bufferi. Na obrázku 3.2 je znázornený presun paketov z DMA bufferov v karte až do *mbufov* v DPDK a naopak.



Obr. 3.2: Znázornenie pohybu paketov medzi DMA bufferami v karte a *mbufmi* v DPDK.

3.3 Merania réžie DPDK a kopírovania dát

Súčasťou DPDK je ovládač virtuálneho zariadenia nazvaný *null*, ktorý umožňuje vytvárať virtuálne zariadenia *eth_null*. Tento ovládač nie je postavený nad žiadnym skutočným sieťo-

vým zariadením. Ovládač simuluje prijímanie a vysielanie prázdnych paketov. Pri vytváraní virtuálneho zariadenia *null* sa zadávajú dva parametre zariadenia:

- „**size**“ – určuje veľkosť prijímaných/vysielaných paketov, všetky pakety prijímané, či vysielané zariadením *eth_null* budú mať rovnakú veľkosť
- „**copy**“ – určuje, či sa budú pri prijímaní/vysielaní paketov dáta kopírovať medzi pomocným bufferom a *mbufom* alebo nie

S použitím ovládača *null* je možné odmerať réžiu prenosov dát v DPDK spojenú s volaním RX a TX funkcií, alokovaním a uvoľňovaním *mbufov*, prípadne kopírovaním dát. K meraniam som použil testovaciu aplikáciu `testpmd`, ktorá je súčasťou DPDK, a verziu DPDK 16.04. Pre merania bol použitý rovnaký stroj ako pre merania výkonnosti reálneho spracovania pomocou karty COMBO-100G. Popis stroja a konfigurácie je v kapitole 5.

Testovacia aplikácia bola spustená príkazom:

```
$ ./testpmd -c 0xfffffffff -n 4 --master-lcore 38 \
    --vdev "eth_null0,size=$TXPKTS,copy=$COPY" \
    -- --port-topology=chained --no-flush-rx --burst=$BURST \
    --total-num-mbufs=65535 \
    --forward-mode=$MODE --rxq=$QUEUES --txq=$QUEUES \
    --coremask=$COREMASK --txpkts=$TXPKTS
```

kde:

- `TXPKTS` určuje veľkosť paketu bez kontrolného súčtu CRC, čiže pre simuláciu prijímania/vysielania po Ethernete bola použitá hodnota veľkosti paketu zmenšenej o 4 B.
- `COPY` určuje, či sa má použiť kopírovanie (hodnota `1`) alebo nie (hodnota `0`).
- `BURST` určuje veľkosť zhlukov paketov.
- `MODE` určuje režim spracovania.
- `QUEUES` určuje počet RX/TX front.
- `COREMASK` určuje rozloženie spracovacích front na jednotlivé jadrá procesoru. Zadaná hodnota je maska, kde „1“ znamená, že logické jadro s daným poradovým číslom je použité na spracovanie paketov.

Merania boli prevedené pre nasledujúce režimy spracovania:

- príjem dát – iba RX smer (`rxonly`)
- vysielanie dát – iba TX smer (`txonly`)
- preposielanie dát – RX aj TX smer (`io`)

Pre každý režim boli merané hodnoty s viacerými konfiguráciami počtu použitých jadier CPU a RX/TX front. Konfigurácie sú uvedené v tabuľke 3.1.

Pre každú konfiguráciu boli merané hodnoty počtu spracovaných paketov za sekundu (pps[1]) a počtu cyklov CPU potrebných na spracovanie jedného paketu. Použité boli paketové dĺžky 64, 128, 256, 512, 1024, 1280 a 1518 B vrátane CRC. Čiže hodnoty `TXPKTS` boli oproti uvedeným dĺžkam zmenšené o 4. Veľkosť zhlukov paketov bola 32.

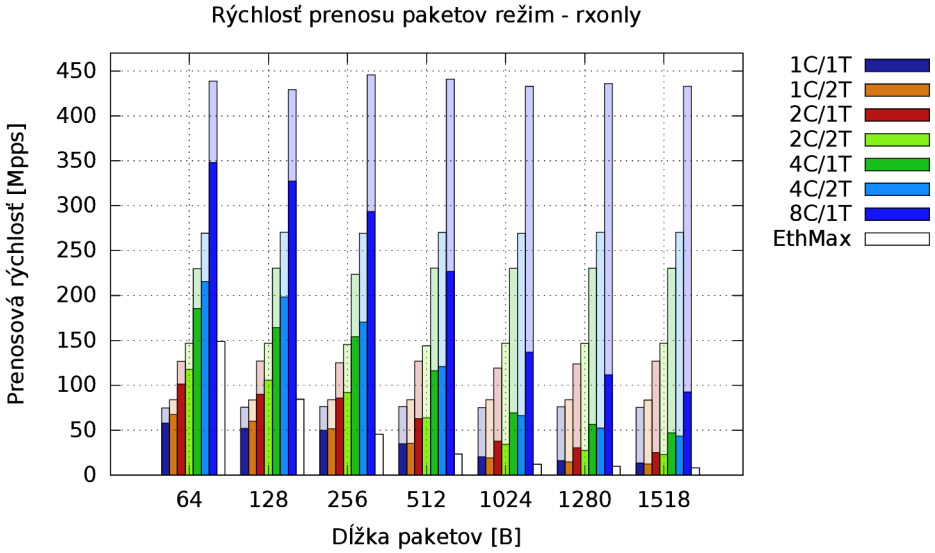
[1] skratka z anglického *packet per second*

Označenie	Fyzické jadrá	Logické jadrá	Fronty	COREMASK	QUEUES
1C/1T	1	1	1	0x0000000001	1
1C/2T	1	2	2	0x0000100001	2
2C/1T	2	2	2	0x0000000005	2
2C/2T	2	4	4	0x0000500005	4
4C/1T	4	4	4	0x0000000055	4
4C/2T	4	8	8	0x0005500055	8
8C/1T	8	8	8	0x0000005555	8

Tabuľka 3.1: Prehľad konfigurácií použitých jadier CPU a RX/TX front.

3.3.1 Príjem dát

Pre tento režim bol nastavený parameter `--forward-mode=rxonly`. V režime čistého prijímania dát treba počítať s tým, že je potrebné aj uvoľňovať mbufy s prijatými paketmi, aby ich bolo možné recyklovať. Počty cyklov CPU na jeden paket boli merané čisto len pre RX funkciu. Nebola rátaná prebytočná réžia spojená s uvoľňovaním prijatých *mbufov*.

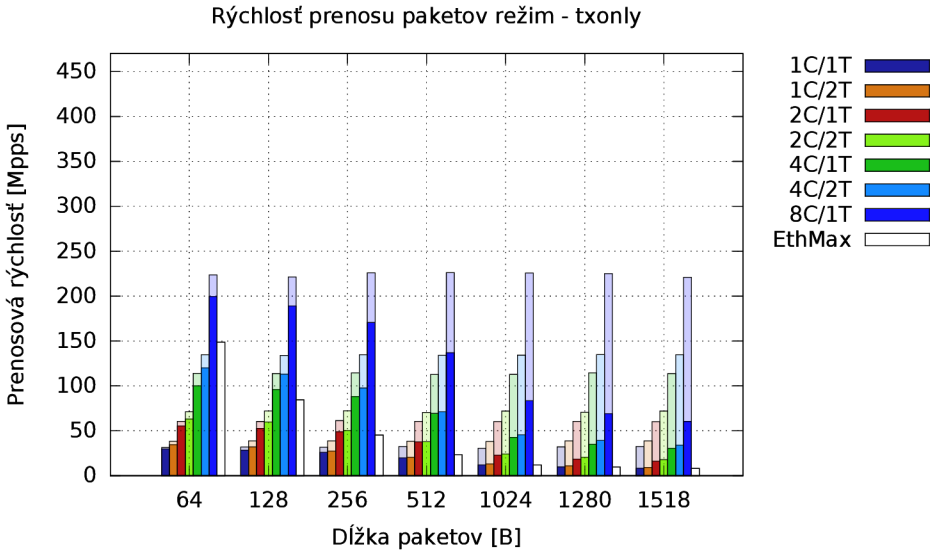


Obr. 3.3: Paketová rýchlosť – režim `rxonly`, veľkosť zhlukov paketov 32.

Na obrázku 3.3 je graf počtu prijatých paketov pre jednotlivé paketové dĺžky s rôznymi počtami použitých jadier CPU a front. *EthMax* je maximálna hodnota, akú je možné dosiahnuť pre konkrétnu paketovú dĺžku na 100G Ethernete. Označenia ostatných stĺpcov sú vysvetlené v tabuľke 3.1. Tmavý stĺpec sú hodnoty s použitím kopírovania dát z pomocného bufferu do *mbufu*. Svetlejší stĺpec sú hodnoty dosiahnuté bez kopírovania dát. Graf predstavuje teoretické limity, ktorým sa snaží ovládač pre COMBO karty priblížiť. Z grafu je zrejmé, že pre prijímanie dát na plnej rýchlosti linky 100G Ethernetu s najkratšími paketmi sú potrebné minimálne 4 jadrá, bez ohľadu na to, či sú dáta do *mbufu* kopírované alebo nie.

3.3.2 Vysielanie dát

Pre tento režim bol nastavený parameter `--forward-mode=txonly`. V tomto režime je pred odoslaním paketov potrebné pripraviť *mbufy*. Testovacia aplikácia okrem alokovania *mbufu* aj vypĺňa hlavičky paketu. Čas potrebný pre vyplnenie hlavičiek paketov nie je zanedbateľný. Preto sú hodnoty v TX smere pre krátke pakety nižšie ako v režime preposielania, kde sú pred odoslaním pakety prijaté RX funkciou, ktorá zaberá menej procesorového času ako príprava paketov pre režim čistého vysielania dát. Počty cyklov CPU na jeden paket boli merané čisto len pre TX funkciu. Nebola rátaná prebytočná réžia alokovania *mbufu* a prípravy hlavičiek paketu pred volaním funkcie na odoslanie paketov.



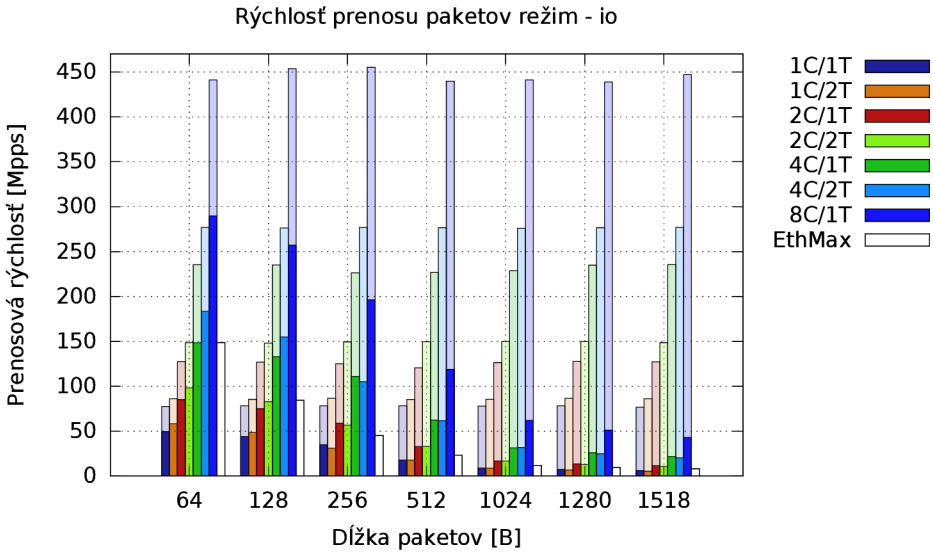
Obr. 3.4: Paketová rýchlosť – režim `txonly`, veľkosť zhlukov paketov 32.

Na obrázku 3.4 je graf počtu odoslaných paketov pre jednotlivé paketové dĺžky s rôznymi počtami použitých jadier CPU a front. *EthMax* je maximálna hodnota, akú je možné dosiahnuť pre konkrétnu paketovú dĺžku na 100G Ethernete. Označenia ostatných stĺpcov sú vysvetlené v tabuľke 3.1. Tmavý stĺpec sú hodnoty s použitím kopírovania dát z *mbufu* do pomocného bufferu. Svetlejší stĺpec sú hodnoty dosiahnuté bez kopírovania dát. Z grafu je zrejmé, že pre dosiahnutie plnej rýchlosti linky 100G Ethernetu s najkratšími paketmi je potrebných až 8 jadier CPU, bez ohľadu na kopírovanie dát. Výrazný podiel na tom má príprava hlavičiek paketu do *mbufu*. Toto je síce umelý prípad, ale treba počítať s tým, že podobné operácie sa môžu vykonávať aj v reálnej aplikácii.

3.3.3 Preposielanie dát

Pre tento režim bol nastavený parameter `--forward-mode=io`. Tento režim najpresnejšie vystihuje samotnú réžiu RX a TX funkcií. Pakety boli prijaté pomocou RX funkcie a ihned odoslané pomocou TX funkcie.

Na obrázku 3.5 je graf počtu preposlaných paketov pre jednotlivé paketové dĺžky s rôznymi počtami použitých jadier CPU a front. *EthMax* je maximálna hodnota, akú je možné dosiahnuť pre konkrétnu paketovú dĺžku na 100G Ethernete. Označenia ostatných stĺpcov sú vysvetlené v tabuľke 3.1. Tmavý stĺpec sú hodnoty s použitím kopírovania dát medzi

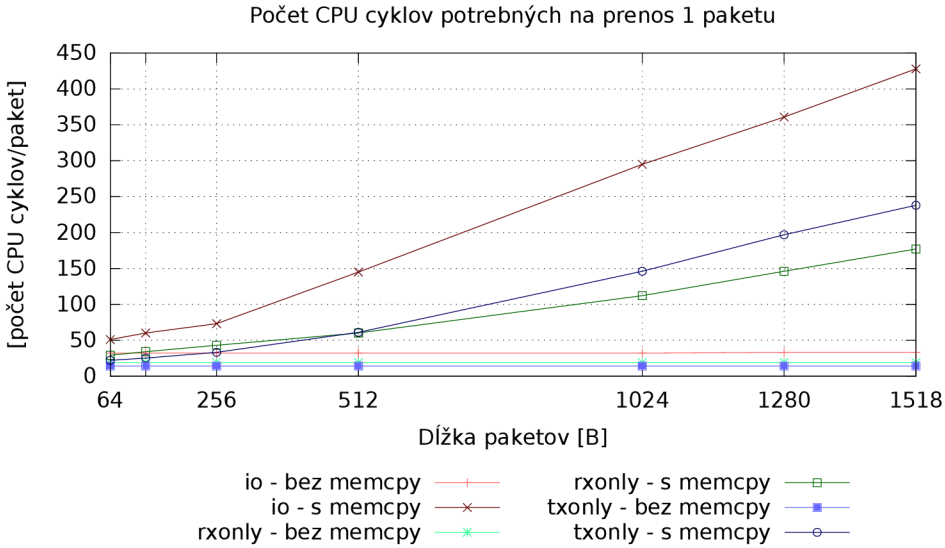


Obr. 3.5: Paketová rýchlosť – režim `io`, veľkosť zhlukov paketov 32.

mbufmi a pomocnými bufferami. Svetlejší stĺpec sú hodnoty dosiahnuté bez kopírovania dát. Z grafu je vidieť, že minimálne 4 jadrá CPU sú potrebné na zvládnutie preposielania najkratších paketov pri plnom zaťažení linky o rýchlosti 100 Gb/s.

3.3.4 Počet cyklov CPU na prenos paketu

Graf na obrázku 3.6 znázorňuje počet cyklov CPU potrebných pre spracovanie jedného paketu vo všetkých troch vyššie uvedených režimoch (`rxonly`, `txonly`, `io`) pri použití 1 jadra s 1 frontou (v každom smere pre režim preposielania).



Obr. 3.6: Počet cyklov CPU na 1 paket – režimy `rxonly`, `txonly`, `io`; 1 fronta.

Kapitola 4

Popis implementácie ovládača szedata2 pre DPDK

V tejto kapitole je popísaná implementácia DPDK ovládača *szedata2* pre sieťové karty COMBO. Ovládač je závislý na knižnici *libsze2*, ktorej API používa pre zabezpečenie DMA prenosov z karty do pamäti RAM v počítači. Ďalšími závislosťami sú moduly jadra: *combov3*, *combo6core*, *szedata2* a *szedata2_cv3*. V sekcii 4.1 je popísaný priebeh vývoja DPDK ovládača *szedata2*. Sekcia 4.2 popisuje funkcie poskytované ovládačom.

4.1 Vývoj ovládača

V oddiele 2.6.6 bolo spomenuté, že ovládač sieťového zariadenia môže byť jedným z dvoch typov: `PMD_VDEV` (virtuálne zariadenie) a `PMD_PDEV` (fyzické zariadenie). Virtuálne zariadenie sa vytvára pomocou argumentu príkazového riadku `--vdev` pri spustení DPDK aplikácie. Tento parameter umožňuje zadávať parametre pre inicializáciu zariadenia. Fyzické zariadenie riadené ovládačom typu `PMD_PDEV` nemá túto možnosť zadania parametrov pre inicializáciu. Funkcia `szedata_open()`, ktorá zabezpečuje inicializáciu v knižnici *libsze2*, potrebuje poznať cestu k súboru znakového zariadenia. Ak by bolo v systéme zapojených viac kariet COMBO, každá z nich by mala vytvorený vlastný súbor */dev/szedataIIX*, kde *X* by označovalo číslo karty. Počas funkcie pre inicializáciu zariadenia v DPDK ovládači je teda potrebné jednoznačne určiť, ktorý súbor patrí k danému zariadeniu.

V prvej verzii bol ovládač *szedata2* implementovaný ako ovládač typu `PMD_VDEV`. Cesta k súboru znakového zariadenia sa zadávala ako parameter pri vytváraní virtuálneho zariadenia. Minimálna potrebná verzia modulov jadra bola 0.9.2. Tento ovládač bol poslaný v podobe záplat do vývojárskej emailovej konferencie DPDK. Ovládač bol prijatý do hlavného stromu DPDK a stal sa súčasťou vydania verzie 2.2.0 (december 2015). Dokumentácia k ovládaču, ktorá je dostupná zo sekcie *Documentation*[1] na oficiálnej stránke projektu DPDK, obsahuje popis a príklad parametrov pre vytvorenie zariadenia pomocou EAL argumentu `--vdev`. Kedže ovládač obsahuje externé závislosti, ktoré nie sú súčasťou DPDK, preklad ovládača je pre východzie nastavenie deaktivovaný. Aktivácia prekladu ovládača je možná nastavením možnosti `CONFIG_RTE_LIBRTE_PMD_SZEDATA2=y` v konfiguračnom súbore. Toto konfiguračné nastaveni ostalo aj pre súčasnú verziu ovládača, ktorá je popisovaná v ďalšom texte.

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

V rámci diskusie o prijatí ovládača do verzie 2.2.0 bolo odporučené prerobiť ovládač na typ `PMD_PDEV`. Tento typ ovládača obsahuje vo svojich interných štruktúrach identifikáciu zariadenia na zbernici PCI-Express. Podľa tejto identifikácie bolo potrebné určiť cestu k súboru */dev/szedataIIX*. SZE2 moduly jadra v tom čase neposkytovali možnosť ako túto cestu získať podľa identifikácie zariadenia na zbernici. Z tohto dôvodu boli moduly upravené[2] tak, aby pridávali do štruktúry súborového systému sysfs špeciálny súbor */sys/class/combo/combosixX/device/pcislot*, kde *X* je číslo karty. Tento súbor obsahuje identifikáciu zariadenia na zbernici. DPDK ovládač prechádza všetky tieto súbory a porovnáva svoju identifikáciu s identifikáciou v súbore. Keď nájde zhodu, číslo *X* je dosadené do cesty k súboru */dev/szedataIIX*. Podpora tejto vlastnosti je v moduloch od verzie 0.9.4. Ovládač prerobený na typ `PMD_PDEV` sa stal súčasťou vydania DPDK verzie 16.04. V tejto verzii je COMBO karta automaticky rozoznaná počas inicializačnej fázy vrstvy EAL a už sa nepoužíva argument `--vdev`.

4.2 Funkcionalita ovládača

API pre DPDK ovládače poskytuje okrem prijímania a odosielania dát aj mnoho funkcií pre konfiguráciu, riadenie karty, filtrovanie dát, zobrazovanie informácií a štatistík. Ovládač však nemusí implementovať všetky funkcie. Podporované funkcie ovládača sú aplikácii sprístupnené pomocou funkčných ukazovateľov v štruktúre `struct eth_dev_ops`. Ak ovládač niektoré funkcie nepodporuje, stačí keď má príslušný ukazovateľ nastavený na `NULL`. Ovládač *szedata2* vo vydaní DPDK 2.2.0 obsahoval iba základné funkcie nevyhnutné pre inicializáciu, spustenie zariadenia a prenosy v RX a TX smere. Do ovládača v nasledujúcom vydaní DPDK bola pridaná podpora niektorých ďalších funkcií:

- `eth_dev_set_link_up()`, `eth_dev_set_link_down()` – aktivácia, resp. deaktivácia komponentov IBUF a OBUF
- `eth_link_update()` – zistenie stavu linky
- `eth_promiscuous_enable()`, `eth_promiscuous_disable()` – zapnutie, resp. vypnutie promiskuitného režimu
- `eth_allmulticast_enable()`, `eth_allmulticast_disable()` – zapnutie, resp. vypnutie prijímania všetkých rámcov typu multicast

Funkcie pre príjem a vysielanie dát sú popísané v samostatných častiach.

Prijímanie dát Ukazovateľ na funkciu zabezpečujúcu prijímanie dát, `rx_pkt_burst`, sa nachádza priamo v štruktúre `struct rte_eth_dev`, ktorá reprezentuje sieťové zariadenie. Ovládač poskytuje dve funkcie pre príjem paketov. Prvá funkcia prijíma len pakety, ktorých dĺžka nepresahuje veľkosť jedného *mbufu*. Dlhšie pakety sú zahadzované. Druhá funkcia využíva reťazenie *mbufov*, takže dokáže prijímať ľubovoľne dlhé pakety. Počas inicializácie a konfigurácie je zvolená funkcia, ktorá sa následne používa pre príjem paketov. Úlohou funkcie je naplniť pole ukazovateľov na *mbufy* ukazovateľmi na *mbufy* s novými paketmi. Počet paketov, ktoré majú byť prijaté je určený parametrom funkcie. Počet skutočne prijatých paketov ja vrátený v návratovej hodnote.

[2] úpravu modulov neimplementoval autor práce

Funkcia `szedata_rx_lock_data()` je použitá na získanie oblasti s novými paketmi. Následne je pre každý paket analyzovaná SZE2 hlavička, z ktorej je zistená dĺžka paketu. Do *mbufu* sú vyplnené potrebné metadáta a skopírované paketové dáta. Pre kopírovanie je použitá funkcia `rte_memcpy()`. Keď sú z danej oblasti presunuté všetky pakety, je zavolaná funkcia `szedata_rx_unlock_data()`, ktorá zabezpečí uvoľnenie a znovupoužitie danej oblasti.

Vysielanie dát V štruktúre `struct rte_eth_dev` sa nachádza aj ukazovateľ na funkciu zabezpečujúcu vysielanie dát, `tx_pkt_burst`. Táto funkcia obdrží pole ukazovateľov na *mbufy* a ich počet. Úlohou funkcie je odoslať pakety, ktoré obsahujú. Spracovanie zreťazených aj nezreťazených *mbufov* zabezpečuje jedna funkcia.

Volaním funkcie `szedata_tx_lock_data()` je získaná oblasť v SZE2 bufferi, do ktorej je možné uložiť pakety, ktoré sa majú preniesť. TX funkcia potom postupne vytvára SZE2 hlavičky a kopíruje dáta zo všetkých *mbufov*, ktoré má odoslať, do SZE2 bufferu. Po skopírovaní paketu je príslušný *mbuf* uvoľnený. Keď je zaplnená oblasť v SZE2 bufferi pre odoslanie paketov, zavolá sa funkcia `szedata_tx_unlock_data()`, ktorá zabezpečí, že sú dáta odoslané do karty.

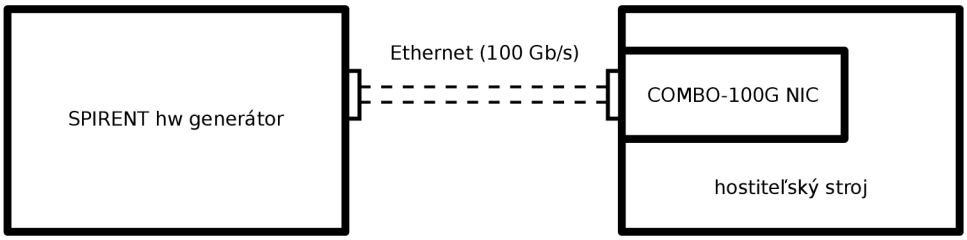
Kapitola 5

Merania výkonnosti DPDK nad kartami COMBO

DPDK ovládač *szedata2* bol otestovaný pomocou aplikácie `testpmd`. Ide o aplikáciu určenú na testovanie funkcionality ovládačov v DPDK. Táto aplikácia bola použitá aj na odmeranie výkonnosti ovládača *szedata2* nad sieťovou kartou COMBO-100G. Aplikácia `testpmd` bola upravená tak, aby umožňovala presné meranie času potrebné pre merania. V tejto kapitole sú popísané prevedené merania a ich výsledky. Merané bolo prijímanie (sekcia 5.3), vysielanie (sekcia 5.4) a preposielanie dát (sekcia 5.5). V sekcii 5.1 je zhrnutá konfigurácia stroja použitého na merania a testovacieho zapojenia. Sekcia 5.2 popisuje prípravu pred spustením a spustenie testovacej aplikácie `testpmd`.

5.1 Charakteristika testovacieho zapojenia

Pre merania bola použitá sieťová karta COMBO-100G a hardvérový generátor sieťových tokov Spirent. Stroj s kartou COMBO-100G bol prepojený so Spirentom linkou o rýchlosti 100 Gb/s. Obrázok 5.1 zobrazuje schému testovacieho zapojenia.



Obr. 5.1: Znázornenie testovacieho zapojenia.

V tabuľke 5.1 sú zhrnuté informácie o použitom testovacom stroji, softvéri a firmvéri pre kartu COMBO-100G.

5.2 Príprava a spustenie testovacej aplikácie

Príprava pred začiatkom meraní Pred spustením meraní s DPDK aplikáciou bolo potrebné vykonať nasledujúce kroky:

1. Nahrať SZE2 moduly jadra vo verzii 0.9.7.

Informácie o testovacom stroji	
Základná doska	Dell Inc. 0H21J3
CPU	Xeon(R) CPU E5-2660 v3 @ 2,60 GHz
Počet jadier	2x10 (2x20)
Hyper-Threading	áno
RAM	64GB DDR4 @ 2133 MHz

Informácie o OS a použitom softvéri	
OS	Scientific Linux release 6.5 (Carbon)
Kernel	2.6.32-431.1.2.el6.x86_64
Verzia SZE2 kernel modulov	0.9.7
Verzia libsze2	1.1.5
Verzia DPDK	16.04

Informácie o karte COMBO-100G a použitom firmvéri	
Karta	COMBO-100G
Firmvér	HANIC_100G1_LR4
Verzia firmvéru	HANIC 4.1
Počet DMA kanálov RX/TX	8/8

Tabuľka 5.1: Prehľad informácií o testovacom stroji, použitom softvéri a firmvéri.

2. Preložiť DPDK vo verzii 16.04 a zlinkovať ovládač *szedata2* s knižnicou *libsze2* vo verzii 1.1.5. Návod k prekladu DPDK je v prílohe B.

3. Rezervovať obrovské stránky. Použil som 1024 stránok. V prílohe B je návod k rezervácii obrovských stránok.

4. Nahrať do karty COMBO-100G potrebný firmvér. Detaily k verzii firmvéru sú v tabuľke 5.1.

5. Aktivovať vstupné (IBUF) a výstupné (OBUF) sieťové rozhrania v karte pomocou nástrojov `ibufctl` a `obufctl`:

   ```
   $ ibufctl -Ae1
   $ obufctl -Ae1
   ```

6. Nastaviť *round-robin* režim distribúcie prijímaných dát medzi DMA kanály pomocou nástroja `hanicctl`:

   ```
   $ hanicctl -c 1
   ```

Priebeh meraní Pred každým spustením testovacej DPDK aplikácie bolo potrebné nastaviť správny počet RX DMA kanálov pomocou nástroja `filterctl`:

```
$ echo "default allow 0-$CHANNELS" > ./filterfile.txt
$ filterctl -F ./filterfile.txt
```

kde `$CHANNELS` reprezentuje počet RX DMA kanálov zmenšený o 1 (napríklad pre použitie 4 kanálov má `$CHANNELS` hodnotu 3).

Testovacia aplikácia `testpmd` bola spustená podobne ako pre merania s ovládačom *null*, ktoré sú popísané v sekcii 3.3. Rozdiel bol iba v tom, že nebol použitý EAL argument `--vdev`. Pre merania boli použité dĺžky paketov tak, ako navrhuje RFC 2544 [2]: 64 B, 128 B, 256 B, 512 B, 1024 B, 1280 B a 1518 B. Do paketovej dĺžky je zarátané aj CRC. Pre každý režim boli merané hodnoty s rovnakými konfiguráciami počtu použitých jadier CPU a RX/TX front ako pre merania s ovládačom *null*. Konfigurácie sú zosumarizované v tabuľke 3.1. RFC 2889 [8] odporúča pre merania rýchlostí preposielania dát[1] dobu 30 s pre každé prevedenie merania, preto som použil túto hodnotu. Pre každú konfiguráciu som meral počet spracovaných paketov za sekundu (pps) a počet cyklov CPU potrebných na spracovanie jedného paketu.

5.3 Prijímanie dát sieťovou kartou COMBO-100G

Aplikácia `testpmd` bola spustená v režime prijímania dát – `rxonly`. Hardvérový generátor Spirent bol použitý pre generovanie konštantného sieťového toku s maximálnym možným množstvom paketov danej dĺžky pre linku o rýchlosti 100 Gb/s. Maximálne rýchlosti prenosu paketov za sekundu pre použité paketové dĺžky sú v tabuľke 5.2. Pomocou nástroja `ibufctl`

Veľkosť paketu [B]	**Paketová rýchlosť [pps]**
64	148809523
128	84459459
256	45289855
512	23496240
1024	11973180
1280	9615384
1518	8127438

Tabuľka 5.2: Maximálne paketové rýchlosti pre 100G Ethernet.

som odmeral počet paketov, ktoré dorazili na vstupné rozhranie, a počet paketov, ktoré boli úspešne presunuté do SZE2 bufferu. Na základe hodnoty počtu paketov prijatých na vstupné rozhranie a maximálnej paketovej rýchlosti pre danú paketovú dĺžku bola vypočítaná doba prijímania dát. Z doby prijímania dát a počtu paketov spracovaných SZE2 bufferom bola vypočítaná rýchlosť prijímania paketov. Počet cyklov CPU potrebných na prijatie jedného paketu bol počítaný aplikáciou `testpmd`.

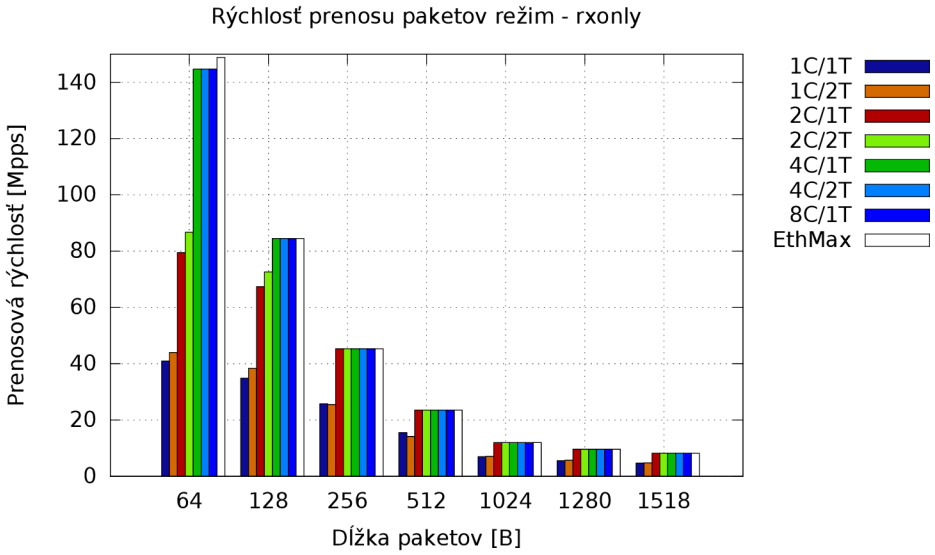
Okrem meraní s ovládačom *szedata2* som odmeral aj výkonnosť modifikácie ovládača, pri ktorej bolo vynechané kopírovanie dát medzi SZE2 bufferom a *mbufom*. To znamená, že prebehli DMA prenosy, metadáta v *mbufoch* boli aktualizované, ale neboli skopírované reálne dáta paketov. Tento experiment mal ukázať úroveň degradácie výkonnosti spôsobenej kopírovaním.

5.3.1 Vyhodnotenie výsledkov

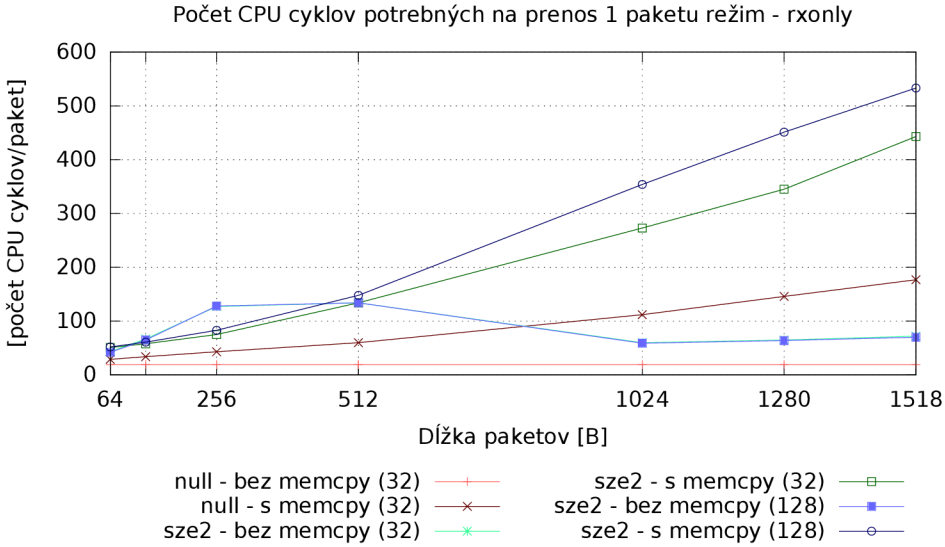
Graf na obrázku 5.2 zobrazuje rýchlosti prijímania paketov pre jednotlivé paketové dĺžky s konfiguráciami z tabuľky 3.1. *EthMax* predstavuje maximálnu paketovú rýchlosť pre danú dĺžku na linke s rýchlosťou prenosu dát 100 Gb/s. Veľkosť zhlukov paketov bola 128. Pre

[1] *Forwarding Rate*

dĺžku paketov 64 B nebola dosiahnutá plná rýchlosť linky. Pre konfigurácie so 4 aj 8 RX frontami bolo dosiahnutých 97 % rýchlosti linky. Paketová dĺžka 128 B už umožňuje prijímať pakety na plnej rýchlosti linky a dostačujúce sú 4 RX fronty. Od paketovej dĺžky 256 B bola dosiahnutá maximálna rýchlosť linky s 2 RX frontami.

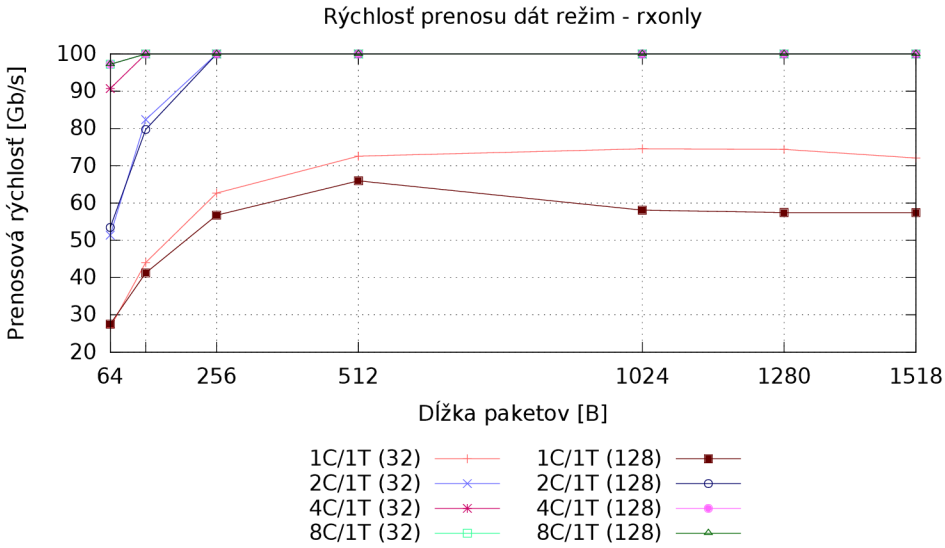


Obr. 5.2: Paketová rýchlosť – režim `rxonly`, veľkosť zhluku paketov 128.



Obr. 5.3: Počet cyklov CPU na 1 paket – režim `rxonly`, 1 RX fronta.

V grafe na obrázku 5.3 je porovnanie počtu cyklov CPU potrebných na prijatie 1 paketu s 1 RX frontou pre ovládač *null* s kopírovaním medzi *mbufom* a pomocným bufferom a bez kopírovania, ovládač *szedata2* a modifikovaný ovládač *szedata2* bez kopírovania paketových dát. Pre ovládač *null* sú zobrazené hodnoty dosiahnuté pri veľkosti zhluku paketov 32. Pre ovládač *szedata2* sú zobrazené hodnoty pre veľkosti zhlukov 32 a 128. Na prijatie 1



Obr. 5.4: Dátová rýchlosť – režim `rxonly`, veľkosti zhlukov 32 a 128.

paketu s minimálnou dĺžkou pri veľkosti zhluku 128 bolo potrebných iba 52 cyklov CPU. S modifikáciou bez kopírovania to bolo o 10 cyklov menej, čo znamená, že pre najkratšie pakety predstavuje kopírovanie dát približne 20 % z celkového času potrebného na prijatie paketu. Prijatie paketu o veľkosti 1518 B vyžadovalo 533 cyklov. Bez kopírovania to bolo iba 70 cyklov. Kopírovanie v tomto prípade predstavuje až približne 87 % času potrebného na prijatie paketu. Pre paketové dĺžky 128 B a 256 B bola pozorovaná výkonová anomália, kedy bola s modifikáciou ovládača *szedata2* bez kopírovania dát dosiahnutá nižšia priepustnosť ako pre klasický neupravený ovládač. Táto anomália je bližšie popísaná v sekcii 5.6.

Graf na obrázku 5.4 porovnáva rýchlosť prijímania dát pre veľkosti zhlukov 32 a 128. V grafe sú zobrazené iba varianty konfigurácií jadier CPU, kde každé jadro spracovávalo iba jednu frontu. Z grafu je zrejmé, že s rastúcou veľkosťou paketu je výhodnejšie používať menšiu veľkosť zhluku.

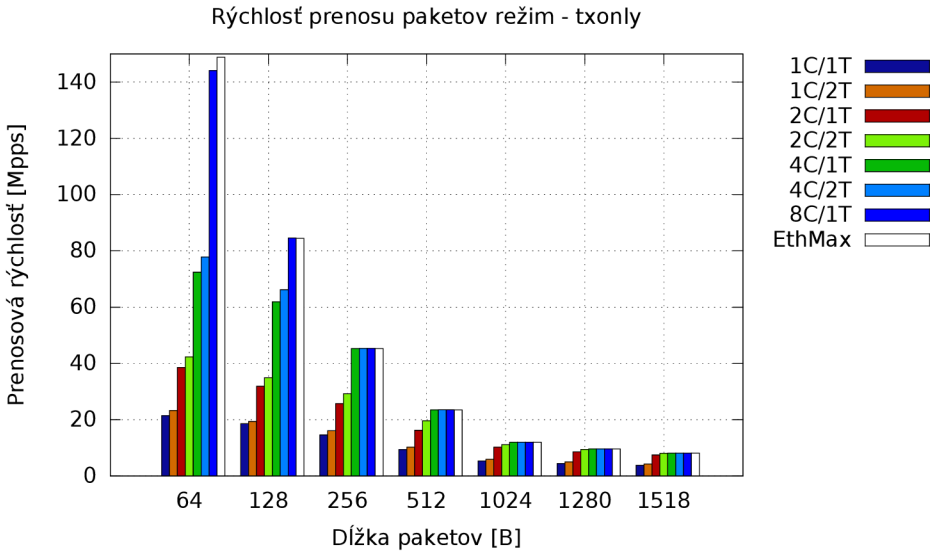
5.4 Vysielanie dát sieťovou kartou COMBO-100G

Pre meranie vysielania dát bola aplikácia `testpmd` spustená v režime `txonly`. Od okamihu spustenia prenosov po zastavenie bol meraný čas a počet odoslaných paketov. Z týchto hodnôt som vypočítal rýchlosť vysielania paketov. Počet cyklov CPU potrebných na odoslanie jedného paketu bol taktiež odmeraný aplikáciou `testpmd`. Rovnako ako pre režim prijímania dát som aj v tomto režime meral výkonnosť ovládača *szedata2* a jeho experimentálnej modifikácie bez kopírovania dát.

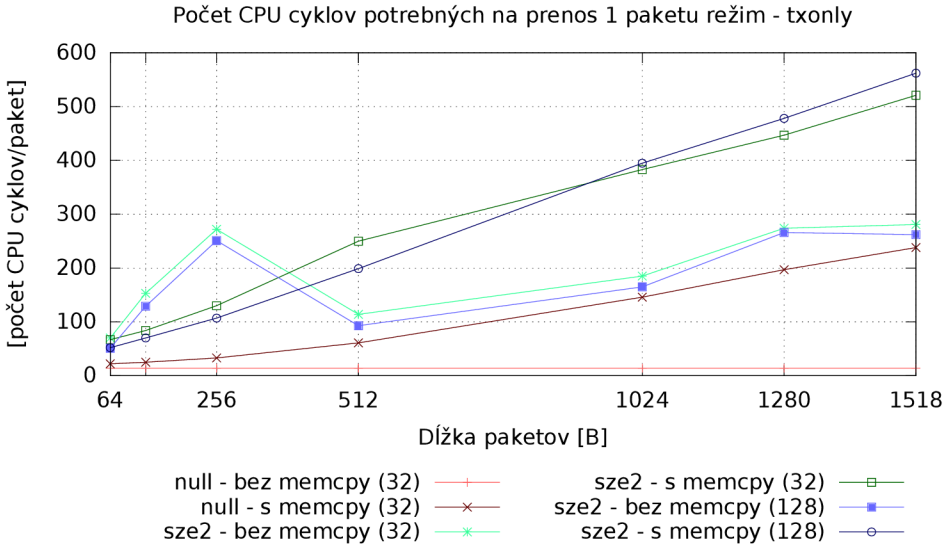
5.4.1 Vyhodnotenie výsledkov

Na obrázku 5.5 je zobrazený graf rýchlosti vysielania paketov pre jednotlivé paketové dĺžky s konfiguráciami podľa tabuľky 3.1. *EthMax* predstavuje maximálnu paketovú rýchlosť pre danú dĺžku na linke s rýchlosťou prenosu dát 100 Gb/s. Použité boli zhluky paketov o veľkosti 128. S najkratšími paketmi bolo možné odoslať až 144 Mpps s využitím 8 jadier CPU a 8 TX front, čo je približne 97 % rýchlosti linky. S paketmi o veľkosti 128 B bola dosiahnutá

plná rýchlosť linky s využitím 8 jadier a od veľkosti 256 B stačili na dosiahnutie rýchlosti linky 4 jadrá.

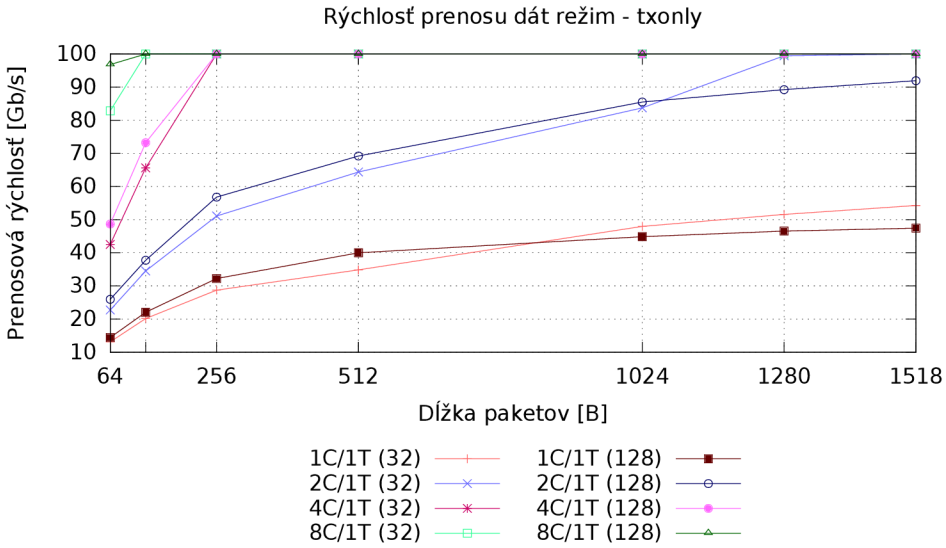


Obr. 5.5: Paketová rýchlosť – režim `txonly`, veľkosť zhluku paketov 128.



Obr. 5.6: Počet cyklov CPU na 1 paket – režim `txonly`, 1 TX fronta.

Rovnako ako v režime `rxonly`, graf na obrázku 5.6 znázorňuje porovnanie počtu cyklov CPU potrebných na odoslanie 1 paketu s 1 TX frontou pre ovládač *null*, *szedata2* a modifikovaný ovládač *szedata2* bez kopírovania. Odoslanie najkratšieho paketu pri veľkosti zhluku 128 zabralo 52 cyklov CPU. Paket o veľkosti 1518 B vyžadoval 562 cyklov. Bez kopírovania to bolo iba 262 cyklov. Kopírovanie v tomto prípade zaberalo približne 53 % času potrebného na odoslanie paketu. Pri odosielaní paketov bola pozorovaná rovnaká anomália



Obr. 5.7: Dátová rýchlosť – režim `txonly`, veľkosti zhlukov 32 a 128.

ako pri prijímaní, kedy pre určité paketové dĺžky bola dosahovaná vyššia priepustnosť pre ovládač bez modifikácií ako pre upravený ovládač bez kopírovania.

Graf na obrázku 5.7 zobrazuje rýchlosť odosielania dát pre veľkosti zhlukov 32 a 128. Z grafu je zrejmé, že pre kratšie pakety je možné dosiahnuť vyššiu priepustnosť s použitím väčších zhlukov.

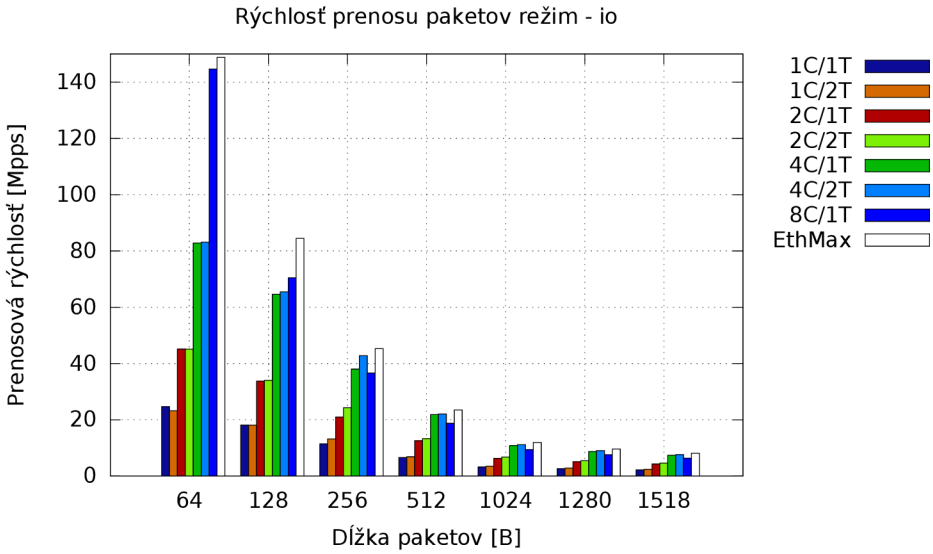
5.5 Preposielanie dát sieťovou kartou COMBO-100G

Východzí režim aplikácie `testpmd` – `io` preposiela dáta prijaté v RX frontách do TX front. Tento režim som použil pre meranie rýchlosti preposielania paketov. Rovnako ako pre merania prijímania dát bol použitý hardvérový generátor Spirent, ktorý generoval konštantný tok dát s maximálnym množstvom paketov danej dĺžky pre linku o rýchlosti 100 Gb/s (tabuľka 5.2). Čítače zo Spirenta boli použité pre meranie množstva odoslaných a prijatých paketov. Z podielu týchto hodnôt a rýchlosti generovania paketov bola vypočítana rýchlosť preposielania paketov. Počet cyklov CPU potrebných na preposlanie jedného paketu bol počítaný aplikáciou `testpmd`. Meral som výkonnosť ovládača *szedata2* a jeho experimentálnej upravenej verzie bez kopírovania dát.

5.5.1 Vyhodnotenie výsledkov

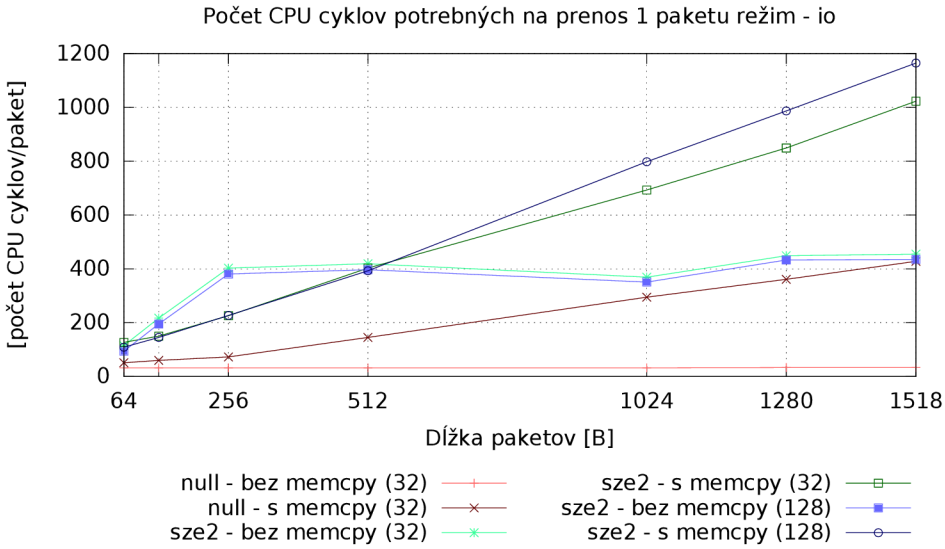
Obrázok 5.8 zobrazuje graf rýchlosti preposielania paketov pre jednotlivé paketové dĺžky s konfiguráciami podľa tabuľky 3.1. *EthMax* predstavuje maximálnu paketovú rýchlosť pre danú dĺžku na linke s rýchlosťou prenosu dát 100 Gb/s. Zhluky paketov obsahovali 128 paketov. Pre paketovú dĺžku 128 B boli dosiahnuté rýchlosti viac ako 80 % rýchlosti linky. Pre všetky ostatné merané paketové dĺžky bolo dosiahnutých dokonca viac ako 90 % rýchlosti linky.

Graf na obrázku 5.9 zobrazuje počet cyklov CPU potrebných na preposlanie jedného paketu s použitím 1 RX a 1 TX fronty pre ovládače *null*, *szedata2* a upravený *szedata2* bez kopírovania dát. Preposlanie najkratšieho paketu pri veľkosti zhlukov 128 zabralo 109



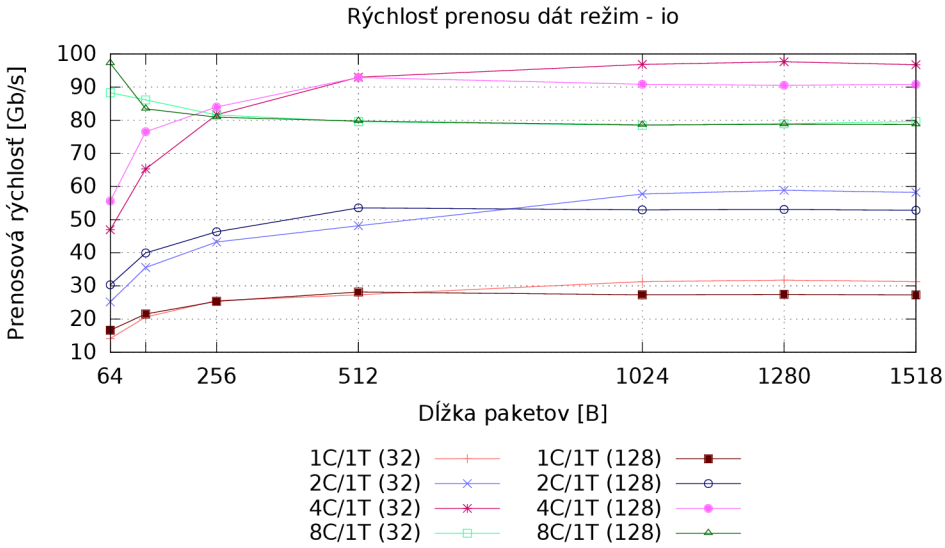
Obr. 5.8: Paketová rýchlosť – režim `io`, veľkosť zhluku paketov 128.

cyklov, čo je o 15 viac ako bez kopírovania. Paket o veľkosti 1518 B bol preposlaný za 1165 cyklov CPU. Bez kopírovania to bolo 435 cyklov. Kopírovanie tak pre najkratšie pakety zabralo približne 14 % a pre najdlhšie pakety približne 63 % celkového času potrebného na preposlanie paketu.



Obr. 5.9: Počet cyklov CPU na 1 paket – režim `io`, 1 RX a 1 TX fronta.

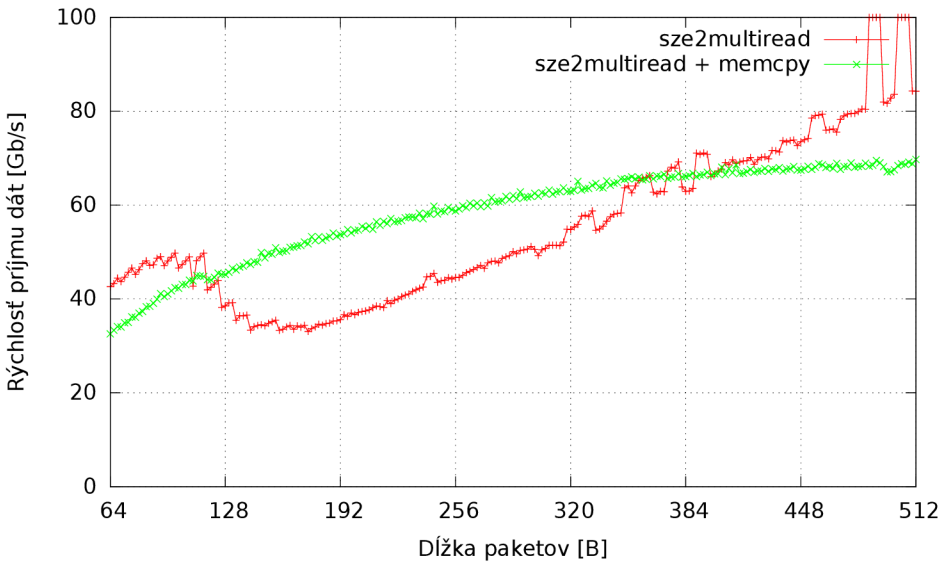
Obrázok 5.10 zobrazuje graf rýchlosti preposielania dát pre veľkosti paketových zhlukov 32 a 128. Pre kratšie dĺžky paketov bola vyššia priepustnosť dosiahnutá pri použití menších zhlukov. Pre dlhšie pakety bola dosahovaná vyššia priepustnosť s väčšími zhlukmi paketov.



Obr. 5.10: Dátová rýchlosť – režim `io`, veľkosti zhlukov 32 a 128.

5.6 Anomálie

Pri meraniach výkonnosti DPDK ovládača *szedata2* bola pre určité paketové dĺžky spozorovaná vyššia priepustnosť prijímania dát v prípade, že boli pakety kopírované zo SZE2 bufferu do *mbufu* v DPDK, ako keď bolo kopírovanie dát vynechané. Táto vlastnosť bola overená aj s použitím existujúceho nástroja *sze2multiread*, ktorý je postavený nad API knižnice *libsze2*. Nástroj prijíma dáta zo všetkých DMA kanálov pomocou funkcie `szedata_read_next()` a počíta počet prijatých paketov a bajtov. Tento nástroj bol pre potreby tohto experimentu upravený tak, aby po prijatí dát skopíroval dáta do statického bufferu. Meral som rýchlosti prijímania dát cez 1 DMA kanál pre paketové dĺžky od 64 do 512 B s krokom 2 B. Namerané hodnoty sú zobrazené v grafe na obrázku 5.11. Z grafu je zrejmé, že pre paketové dĺžky približne od 118 do 390 B je dosahovaná vyššia priepustnosť, keď sú dáta po prijatí do SZE2 bufferu cez DMA prenosy následne kopírované do iného bufferu v pamäti.



Obr. 5.11: Graf rýchlosti príjmu dát v závislosti na paketovej dĺžke s nástrojom sze2multiread. Červenou sú znázornené hodnoty namerané s pôvodným nástrojom. Zelenou sú znázornené hodnoty namerané s upravenou verziou nástroja používajúcou `memcpy()`.

Kapitola 6

Záver

Táto práca popisuje ovládač *szedata2* v aplikačnom rámci DPDK a princípy, na ktorých je založený. Ovládač pridáva podporu sieťových kariet generácie COMBOv3 do DPDK a stal sa súčasťou hlavnej vývojovej línie DPDK. Implementovaný ovládač som testoval a zmeral jeho priepustnosť nad kartou COMBO-100G pomocou programu `testpmd` z prostredia DPDK. V práci sú okrem popisu princípov a implementácie popisané aj prevedené merania a dosiahnuté výsledky.

DPDK ovládač *szedata2* je závislý na knižnici *libsze2* a moduloch jadra, ktoré zabezpečujú DMA prenosy medzi sieťovou kartou a pamäťou RAM v počítači. Hlavným účelom ovládača je kopírovanie paketov medzi SZE2 bufferami a *Message Bufferami* v DPDK. Táto práca poskytuje informácie o tom, prečo je potrebné dáta kopírovať a akú výkonnosť tak bolo možné dosiahnuť.

Meral som výkonnosť prijímania dát (RX), vysielania dát (TX) a preposielania dát s DPDK ovládačom *szedata2* nad kartou COMBO-100G. Ovládač v DPDK je schopný prijímať aj odosielať až 97 % prevádzky na linke o rýchlosti 100 Gb/s na najkratších paketoch (64 B) s využitím 8 jadier CPU. Plnú priepustnosť je možné dosiahnuť s využitím 4 jadier v RX smere a 8 jadier v TX smere na paketoch o dĺžke 128 B. Pre pakety od dĺžky 256 B je možné dosiahnuť plnú rýchlosť linky s využitím len 2 jadier v RX smere a 4 jadier v TX smere. Pri preposielaní paketov je možné dosiahnuť viac ako 80 % rýchlosti linky pre pakety o dĺžke 128 B a pre ostatné merané paketové dĺžky bolo dosiahnutých viac ako 90 % rýchlosti linky.

Na túto prácu je možné nadviazať implementáciou ďalších funkcií pre konfiguráciu karty, využitie filtrovacích schopností karty, hardvérových čítačov a štatistík. Ovládač *szedata2* je závislý na externej knižnici a moduloch jadra, ktoré nie sú súčasťou DPDK. Funkcionalitu modulov jadra a knižnice *libsze2* je možné v ďalšej práci implementovať priamo do ovládača v DPDK a zbaviť sa tak týchto závislostí. Taktiež je možné preskúmať možnosti vytvorenia nového princípu DMA prenosov nad kartami COMBO tak, aby sa dala minimalizovať potreba kopírovania paketov.

Literatúra

[1] Barbette, T.; Soldani, C.; Mathy, L.: Fast Userspace Packet Processing. In *Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '15, Washington, DC, USA: IEEE Computer Society, 2015, ISBN 978-1-4673-6632-8, s. 5–16.
URL http://dl.acm.org/citation.cfm?id=2772722.2772727

[2] Bradner, S.; McQuaid, J.: Benchmarking Methodology for Network Interconnect Devices. RFC 2544 (Informational), Březen 1999, updated by RFCs 6201, 6815.
URL http://www.ietf.org/rfc/rfc2544.txt

[3] Fusco, F.; Deri, L.: High Speed Network Traffic Analysis with Commodity Multi-core Systems. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, New York, NY, USA: ACM, 2010, ISBN 978-1-4503-0483-2, s. 218–224, doi:10.1145/1879141.1879169.
URL http://doi.acm.org/10.1145/1879141.1879169

[4] Gallenmüller, S.; Emmerich, P.; Wohlfart, F.; aj.: 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, Washington, DC, USA: IEEE Computer Society, 2015, ISBN 978-1-4673-6632-8, s. 29–38.
URL http://dl.acm.org/citation.cfm?id=2772722.2772729

[5] García-Dorado, J. L.; Mata, F.; Ramos, J.; aj.: *Data Traffic Monitoring and Analysis: From Measurement, Classification, and Anomaly Detection to Quality of Experience*, kapitola High-Performance Network Traffic Processing Systems Using Commodity Hardware. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, ISBN 978-3-642-36784-7, s. 3–27, doi:10.1007/978-3-642-36784-7_1.
URL http://dx.doi.org/10.1007/978-3-642-36784-7_1

[6] HANK, A.: *Návrh síťových aplikací na platformě NetCOPE [online]*. Diplomová práce, FIT VUT v Brně, 2009 [cit. 2016-01-30].
URL http://www.fit.vutbr.cz/study/DP/DP.php.cs?id=8195

[7] Jacobson, V.; Leres, C.; McCanne, S.: Manpage of PCAP. 2015, [Online; cit. 2016-01-30].
URL http://www.tcpdump.org/manpages/pcap.3pcap.html

[8] Mandeville, R.; Perser, J.: Benchmarking Methodology for LAN Switching Devices. RFC 2889 (Informational), Srpen 2000.
URL http://www.ietf.org/rfc/rfc2889.txt

[9] Rizzo, L.; Deri, L.; Cardigliano, A.: 10 Gbit/s Line Rate Packet Processing Using Commodity Hardware: Survey and new Proposals. [Online; cit. 2016-04-12].
URL http://luca.ntop.org/10g.pdf

[10] SLABÝ, J.: *Rychlé datové přenosy na platformě COMBO [online]*. Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2008 [cit. 2016-01-30].
URL http://is.muni.cz/th/98734/fi_m/

[11] Stevens, R. W.; Fenner, B.; Rudoff, A. M.: *UNIX Network Programming*. Boston: Addison-Wesley, třetí vydání, 2004, ISBN 01-314-1155-1.

[12] Tang, L.; Yan, J.; Sun, Z.; aj.: Towards high-performance packet processing on commodity multi-cores: current issues and future directions. *Science China Information Sciences*, ročník 58, č. 12, 2015: s. 1–16, ISSN 1869-1919, doi:10.1007/s11432-015-5484-6.
URL http://dx.doi.org/10.1007/s11432-015-5484-6

[13] Data Plane Development Kit. [Online; cit. 2016-04-04].
URL http://www.dpdk.org

[14] DPDK Programmer's Guide Release 2.2.0. 2016, [Online; cit. 2016-01-30].
URL http://dpdk.org/doc/pdf-guides/prog_guide-2.2.pdf

[15] Napi. 2009, [Online; cit. 2016-05-01].
URL
http://www.linuxfoundation.org/collaborate/workgroups/networking/napi

[16] Netmap – the fast packet I/O framework. [Online; cit. 2016-05-02].
URL http://info.iet.unipi.it/~luigi/netmap/

[17] Packet(7) – Linux Programmer's Manual. 2015, [Online; cit. 2016-01-30].
URL http://man7.org/linux/man-pages/man7/packet.7.html

[18] PF_RING ZC (Zero Copy). [Online; cit. 2016-05-02].
URL http:
//www.ntop.org/products/packet-capture/pf_ring/pf_ring-zc-zero-copy/

[19] Project Liberouter website – Cards gallery. [Online; cit. 2016-05-02].
URL https://www.liberouter.org/cards-gallery/

[20] Project Liberouter website. [Online; cit. 2016-05-05].
URL https://www.liberouter.org

[21] Raw(7) – Linux Programmer's Manual. 2015, [Online; cit. 2016-01-30].
URL http://man7.org/linux/man-pages/man7/raw.7.html

[22] Tcp(7) – Linux Programmer's Manual. 2015, [Online; cit. 2016-01-30].
URL http://man7.org/linux/man-pages/man7/tcp.7.html

[23] Udp(7) – Linux Programmer's Manual. 2013, [Online; cit. 2016-01-30].
URL http://man7.org/linux/man-pages/man7/udp.7.html

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje nasledujúce adresáre a súbory:

- `bp-xvidom00.pdf`
  Písomná správa vo formáte PDF.
- `manual.txt`
  Návod k použitiu DPDK ovládača *szedata2*.
- `README.txt`
  Súbor s popisom obsahu CD.
- `src/`
  Adresár obsahujúci zdrojové súbory DPDK ovládača *szedata2*.
- `text/`
  Zdrojové texty a obrázky písomnej správy pre LaTeX.

Príloha B

Manuál

Kompletné zdrojové kódy DPDK obsahujúce aj ovládač szedata2 pre COMBO karty je možné nájsť na oficiálnych stránkach DPDK[1]. Stiahnutie vydania DPDK 16.04:

```
$ wget http://dpdk.org/browse/dpdk/snapshot/dpdk-16.04.tar.gz
```

Získanie aktuálneho kódu z gitu:

```
$ git clone http://dpdk.org/git/dpdk
```

Na stránkach DPDK sú aj rýchle inštrukcie[2] pre preklad a konfiguráciu DPDK. V adresári s DPDK je potrebné spustiť konfiguráciu:

```
$ make config T=x86_64-native-linuxapp-gcc
```

V konfiguračnom súbore `./build/.config` je potrebné aktivovať preklad ovládača szedata2 nastavením voľby `CONFIG_RTE_LIBRTE_PMD_SZEDATA2=y`. Pre preklad je potrebné mať v PATH uvedenú cestu k adresáru s nainštalovanou knižnicou libsze2. Následne môžme spustiť preklad:

```
$ make
```

Adresár `./build` obsahuje preložené knižnice a testovacie alikácie DPDK. V adresári `./build/app` je možné nájsť testovaciu aplikáciu `testpmd`.

Pred spustením aplikácie je potrebné rezervovať obrovské stránky:

```
$ mkdir -p /mnt/huge
$ mount -t hugetlbfs nodev /mnt/huge
$ echo 1024 > /sys/devices/system/node/node0/hugepages/hugepages-2048kB/
    nr_hugepages
```

V systéme so zapojenou kartou COMBO-100G a načítanou správnou verziou SZE2 modulov jadra (pre vydanie DPDK 2.2.0 je minimálna verzia 0.9.2, pre vydanie DPDK 16.04 je minimálna verzia 0.9.4) je možné spustiť aplikáciu `testpmd` bežiacu nad kartou COMBO-100G. Napríklad prijímanie paketov z nultého a prvého RX DMA kanálu a preposielanie týchto paketov na nultý a prvý DMA kanál v TX smere je možné príkazom:

```
$ ./build/app/testpmd -c 0xf -n 2 -- -i -a \
--port-topology=chained --no-flush-rx \
--rxq=2 --txq=2 --nb-cores=2
```

Ďalšie informácie je možné nájsť aj v aktuálnej dokumentácii pre ovládač szedata2[3].

[1] http://dpdk.org/
[2] http://dpdk.org/doc/quick-start
[3] http://dpdk.org/doc/guides/nics/szedata2.html