VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

NÍZKOLATENČNÍ KOMUNIKACE MEZI AKCELERAČNÍ KARTOU A UŽIVATELSKOU APLIKACÍ

A LOW-LATENCY COMMUNICATION BETWEEN AN ACCELERATION CARD AND USER APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **HYNEK ŠABACKÝ**
AUTHOR

VEDOUCÍ PRÁCE **Ing. TOMÁŠ MARTÍNEK, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Šabacký Hynek**
Program: Informační technologie
Název: **Nízkolatenční komunikace mezi akcelerační kartou a uživatelskou aplikací**
A Low-Latency Communication between an Acceleration Card and User Application
Kategorie: Počítačová architektura

Zadání:

1. Seznamte se s technologiemi pro nízkolatenční přenosy dat mezi akcelerační kartou a uživatelskou aplikací jako jsou např. eXpress Data Path (XDP) nebo framework DPDK.
2. Seznamte se s platformou NDK a ovladači akceleračních karet COMBO vyvíjených v rámci sdružení CESNET.
3. Navrhněte novou strukturu ovladačů popř. dalších vrstev software pro akcelerační karty COMBO, které budou minimalizovat latenci komunikace s uživatelskou aplikací.
4. Proveďte implementaci navrženého software a jeho funkčnost ověřte na dostupném hardware.
5. Zhodnoťte dosažené výsledky a diskutujte další pokračování práce.

Literatura:

- Dle pokynů vedoucího práce.

Pro udělení zápočtu za první semestr je požadováno:

- Splnění bodů 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Martínek Tomáš, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 29. října 2021

Abstrakt

Tato práce se zabývá analýzou, návrhem a úpravou ovladače zařízení a knihovny platformy NFB, které společně řídí komunikaci akcelerační karty a uživatelské aplikace. Nejdůležitější části této komunikace jsou DMA přenosy mezi RAM a samotným čipem na kartě. Cílem úpravy je minimalizovat latenci těchto přenosů a režie s tím spojenou.

Abstract

This thesis deals with the analysis, design and modification of the the NFB platform device driver and library, which together control the communication between an acceleration card and user application. The most important parts of this communication are DMA transfers between RAM and the card itself. The goal of the modification is to minimize the latency of these transfers and the overhead associated with them.

Klíčová slova

DMA, ovladač zařízení, Linux, nízkolatenční komunikace, platforma NFB

Keywords

DMA, device driver, Linux, low-latency communication, NFB platform

Citace

ŠABACKÝ, Hynek. *Nízkolatenční komunikace mezi akcelerační kartou a uživatelskou aplikací*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Martínek, Ph.D.

Nízkolatenční komunikace mezi akcelerační kartou a uživatelskou aplikací

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Martínka, Ph.D. Další informace mi poskytl Ing. Martin Špinler. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Hynek Šabacký
8. května 2022

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Tomáši Martínkovi, Ph.D. za jeho pomoc a trpělivost při tvorbě této práce. Zároveň bych chtěl poděkovat Ing. Martinu Špinlerovi za odborné konzultace a vstřícnou komunikaci. V neposlední řadě děkuji projektu Liberouter CESNET za poskytnutí prostředí a nástrojů pro vytvoření této práce.

Obsah

Kapitola 1

Úvod

V posledních letech stále roste poptávka po vyšších propustnostech a nižších latencích u síťových provozů. Velikost a hlavně množství dat putujících po Internetu neustále roste a je třeba, aby se na to zařízení adaptovala. Nejvíce tyto nárůsty pociťují uzly, skrze které provoz putuje a je přesměrováván. Tyto uzly často ale i zaznamenávají informace o procházejících datech, nebo na ně nahlíží z například bezpečnostního hlediska. Proto je důležité, aby samotný tok dat byl co nejméně zpomalován pohybem v síti. Pohyb v síti je poměrně široký pojem a obsahuje několik dílčích částí. Výsledná rychlost celého přenosu se odvíjí od součtu všech článků tohoto pomyslného řetězce a je důležité, aby každý z nich zpomaloval provoz co nejméně. Putování paketu z aplikace jednoho počítače do aplikace druhého počítače se dá zjednodušit na tři části. Cesta z aplikace do síťové karty, přesun mezi kartami po síti a průchod přes síťovou kartu do aplikace. Tato práce se zaměřuje na první a poslední krok této transakce.

Pro dosažení nejnižších latencí se využívá hardwarové akcelerace. Sdružení CESNET se zabývá vývojem akceleračních karet pro tyto účely. Jsou to síťové karty, které využívají programovatelných hradlových polí. Tyto pole lze nakonfigurovat uživatelem dle speciálních potřeb, a tím dosáhnout různých funkcionalit. Takto nastavená karta pak využívá naprogramované komponenty pro přenos dat z aplikace do sítě. Aby si ale karta rozuměla se zařízením, ke kterému je připojena, musí být ovládána pomocí ovladače zařízení. To je část jádra operačního systému, která má přístup k paměťovému prostoru karty a může ji ovládat. Každý hardware má svůj vlastní ovladač, který pro uživatelské aplikace standardizuje práci se zařízením. V případě těchto karet zprostředkovává i předávání dat mezi aplikací a rozhraním karty.

Cílem této práce je analyzovat a optimalizovat aktuální implementaci knihovny a ovladače, kterými je ovládána komunikace mezi kartami COMBO a uživatelskou aplikací. Optimalizací je v tomto případě myšlena minimalizace latence při datových přenosech. Pro tyto karty aktuálně existují dvě různé implementace Direct Memory Access modulů a tato práce se vztahuje na obě. Protože je pozůstatek využíván na více místech, je součástí zadání i zpětná kompatibilita nového návrhu. Aplikace by měly fungovat stále stejně, ale vnitřní struktura pozůstatku může být modifikována.

V následující kapitole 2 budou popsány technologie související s touto problematikou. Počínaje obecnými principy Linuxových operačních systémů. Následují DMA moduly, které jsou hojně využívané pro hardwarovou akceleraci datových přenosů v rámci systému. Dále budou popsány platformy NDK a NFB. Obě jsou vyvíjené v rámci CESNETu a spolu zprostředkovávají síťový provoz. Obecně NDK obsahuje firmware pro FPGA akcelerační karty a NFB software pro operační systém zařízení, ke kterému je karta připojena. Na

závěr této kapitoly jsou ještě popsány alternativní softwarové způsoby, kterými lze pakety přenášet. Kapitola 3 navazuje na zmíněné technologie a dívá se na ně z kritického hlediska.

Pozůstatek teoretické a praktické části zahajuje kapitola 4. Nejdříve je pohled na motivaci práce, následně se hledá příčina latence v aktuálním návrhu a nakonec jsou řešeny identifikované problémy. V kapitole 5 je řešení těchto problémů rozvedeno a to pro oba současné moduly DMA. Jedná se tu o modul SZE a modul Medusa. Nakonec je v kapitole 6 vysvětleno, jak byla aplikace testována a měřena. Nejdříve je přiblížena již existující aplikace používaná pro testování. Je ukázáno jakým způsobem byla tato aplikace upravena, aby bylo možné pomocí ní provádět i měření. Práci uzavírá zobrazení dosažených výsledků práce a následně jejich zhodnocení.

Kapitola 2

Technologie

2.1 Linuxové jádro a ovladače zařízení

Jedna z nejkritičtějších komponent Linuxových operačních systémů je Linuxové jádro (kernel). Původně bylo vyvíjeno Linusem Torvaldsem od roku 1991 a jeho hlavní podstatou i podstatou celého operačního systému GNU/Linux je svoboda a otevřenost. Díky tomu se v současné době na vývoji podílí velká řada jednotlivců, ale i korporací, jako jsou například Red Hat, Intel, IBM, SUSE a jiné. Existuje velké množství Linuxových systémů neboli distribucí, jako jsou Ubuntu, Arch Linux, Debian, Fedora a jiné [9, 10]. V rámci práce byla cílová distribuce Fedora, a proto když bude nadále zmíněn operační systém, bude řeč právě o ní.

Linuxové jádro (dále označováno jen jako jádro nebo kernel) slouží jako prostředník mezi hardwarovými komponentami a aplikačními procesy. Jeho hlavním cílem je spravovat paměť a procesy, obsluhovat systémové volání procesů pomocí SCI (system call interface) a zpřístupňovat hardware využitím ovladačů zařízení (device drivers). Jádro jako takové je pro uživatelské aplikace odstíněné. Má vlastní operační paměť a běží v tzv. kernel módu. Kromě kernel módu existuje i user mód. User mód na rozdíl od kernel módu neumožňuje procesům přístup přímo k hardwaru. Proto když proces potřebuje například alokovat paměť nebo komunikovat s hardwarovou komponentou, musí požádat jádro pomocí systémového volání. Při tomto volání se provede přepnutí kontextu privilegií z user režimu na kernel mód. Dále se provedou potřebné funkce požadovaného volání a dojde k následnému zpětnému přepnutí kontextu a navrácení kontroly původnímu procesu. Mezi tyto volání konkrétně patří například `mmap`, nebo `ioctl`. Volání `mmap` slouží pro mapování adresního prostoru zařízení do operační paměti. Volání `ioctl` poskytuje aplikaci způsob, kterým lze komunikovat se zařízením. V rámci tohoto volání se zasílá i identifikační kód, podle kterého ovladač zjistí, o jakou operaci aplikace žádá [1, 8].

Stejným způsobem jako druhy běhu systému je i operační paměť dělena na kernel space (prostor jádra) a user space (uživatelský prostor). Několik megabytů RAM, obvykle na začátku, je dedikováno pro kernel (např. jeho kód nebo statické datové struktury) a zbytek je použit pro:

- Dodatečné místo pro alokaci kernel space paměti (buffery, dynamické datové struktury a další).
- Místo pro alokaci user space paměti neboli paměti procesů.

- Místo pro zlepšení výkonnosti pevných disků a jiných načtených zařízení pomocí techniky zvané caching (uchovávání kopie dat na místě, které je rychleji dostupné, než místo, na kterém byla data původně).

Tato paměť je řízena systémem virtuální paměti (virtual memory system), jenž mimo jiné balancuje rozložení požadavků [1].

Adresace v operačních systémech je dělena do tří tříd podle typu používané adresy:

- Logická adresa – používaná instrukcemi pro adresaci operandu nebo instrukce.
- Virtuální adresa – celé číslo bez znaménka (unsigned integer), které reprezentuje adresu v rámci virtuální vrstvy.
- Fyzická adresa – unsigned integer udávající fyzickou adresu, která odpovídá interní číselné reprezentaci paměti.

Virtuální adresa je poskytována procesům při virtualizaci paměti. Její hlavní podstatou je abstrakce práce s fyzickou pamětí vytvořením nadřazené virtuální adresové vrstvy (Virtual address space). Tato virtuální vrstva není izomorfní s fyzickou vrstvou. Je kapacitně teoreticky neomezená a slouží ke kompenzaci nedostatku operační paměti. V případě, že není dostatek místa v RAM, se část jejího právě nepoužívaného obsahu odloží na pevný disk (do tzv. paging file) a nově uvolněné místo se zaplní potřebnými daty. Virtuální vrstva zůstane nezměněna a při přístupu na virtuální adresu odložených dat, jsou data opět načtena do operační paměti. Překlad virtuálních adres na adresy fyzické provádí Jednotka správy paměti (MMU – Memory Management Unit) za pomoci jádra. Uživatelské aplikace mají přístup jen k virtuálním adresám a je–li je potřeba přeložit, lze to pouze použitím systémového volání [1, 14].

Jak už bylo zmíněno, jednou z funkcí jádra je zpřístupňování hardwarových komponent pomocí ovladačů zařízení. Lze díky nim komunikovat se vstupně–výstupními (I/O – Input/Output) zařízeními jako jsou například klávesnice, monitory, DMA komponenty a jiné. Ovladače se dají chápat jako podmnožina modulů. Moduly v operačním systému jsou objektové kódy, které se dají k jádru dynamicky přidávat a odebírat použitím linkeru, který spojuje více objektových kódu do jednoho spustitelného kódu, přímo za běhu. Při jejich přidání se volá inicializační funkce modulu a při odpojení zase ukončovací funkce. Moduly se dají rozřadit do nejméně tří základních tříd:

- Znaková zařízení – v literatuře nejčastěji jako character device nebo zkráceně char device, jsou brány jako sekvence dat. Jsou umístěny v souborovém systému stejným způsobem jako soubory. Rozdíl mezi nimi a obyčejnými soubory je, že v souborech se lze pohybovat dopředu a dozadu, ale většinu char zařízení lze číst pouze sekvenčně. Nicméně existují způsoby, kterými jde zajistit i náhodné čtení (`lseek`, `mmap`...). Mezi nejčastější operace používané s těmito zařízeními patří `open`, `close`, `read` a `write`.
- Bloková zařízení – tyto zařízení mohou být adresovány libovolně a mohou hostovat souborový systém, ale jinak mají stejné vlastnosti jako char device. Jsou to například pevné disky, CD-ROM ovladače, DVD přehrávače a jiné.
- Síťové rozhraní – při síťové komunikaci je zapotřebí rozhraní (interface), pomocí kterého lze data šířit mezi ostatní počítače. Tato rozhraní obvykle bývají navázána přímo na fyzické komponenty, ale může se stát, že se jedná o „rozhraní softwarové“, což je skutečnost například pro `looback`. Toto rozhraní je ovládáno síťovým podsystémem

kernelu a i při proudovém přenosu (TCP) s pakety pracuje jednotlivě. Síťová rozhraní nejsou na rozdíl od znakových a blokových zařízení umístěny v souborovém systému. Hlavním důvodem je obtížnost mapování kvůli jejich charakteristice. Proto bývávají různě pojmenovávány (např. `eth0`) a jsou zpřístupňovány pomocí funkcí spojených s paketovým přenosem.

Pro zmíněné zařízení neukazuje i–uzel na blok dat na disku, ale obsahuje identifikátor hardwarového zařízení, jenž tento soubor představuje. Nejčastěji se takovýto identifikátor skládá ze dvou částí: major a minor čísel. Major číslo označuje druh zařízení a minor číslo identifikuje jednotlivé zařízení ve skupině zařízení stejného major čísla. Obvykle soubor zařízení reprezentuje reálnou hardwarovou komponentu nebo její část. Může se ale stát, že pouze reprezentuje abstraktní objekt. Jako příklad může být třeba `/dev/null`, který slouží jako „odpadní roura“. Vše co je do něj zapsáno je smazáno a při čtení se tváří jako prázdný soubor [1, 2].

2.2 Direct Memory Access

Přesouvání dat v rámci síťové komunikace je hojně využívaná operace, která je výpočetně velmi náročná. Při použití techniky zvané Programmed input–output (PIO) je pro každý přenos z operační paměti do periferního zařízení zapotřebí instrukcí řízená činnost procesoru. Kdyby měl být procesor těmito přenosy zatěžován, výrazně by se snížila propustnost zařízení. Proto se k tomuto účelu často používá hardwarová akcelerace, kterou představuje technologie Direct Memory Access (DMA – přímý přístup do paměti) [7]. V rámci firmwarové platformy vyvíjené ve sdružení CESNET, se používají převážně implementace s názvem SZE a MEDUSA, které budou podrobněji popsány v sekcích 2.3.1 a 2.3.2.

Řadič DMA je hardwarová komponenta, která je schopna přenášet data mezi RAM a periferním zařízením. Je k oběma připojena sběrnicí, na kterou dokáže generovat čtecí a zápisové transakce. Pro její využití musí I/O zařízení poslat řadiči požadavek na přenos. Ten zažádá procesor o přístup ke sběrnici. Pokud mu je vyhověno, oznámí to zařízení a provede přenos dat. Po ukončení práce navrátí sběrnici zpět procesoru. Samotný přenos je tedy procesoru odstíněn a ten mezi tím může zpracovávat instrukce, které nevyžadují přistup do paměti.

Přenos dat řadiče DMA je standardně ovládán pomocí zápisu dat do jeho řídících registrů. Mezi ně patří registr, jehož naplněním hodnotou lze řadič spustit, nebo zastavit. Aby řadič věděl s jakou operační pamětí má pracovat, jsou připravené registry na hodnotu začátku fyzické paměti a délku tohoto úseku. Pro přenos mohou být i další hodnoty, které jsou zapisovány do řadiče, ale tyto zmíněné jsou nezbytné. Zápisy jsou nejčastěji prováděny pomocí ovladače zařízení [3].

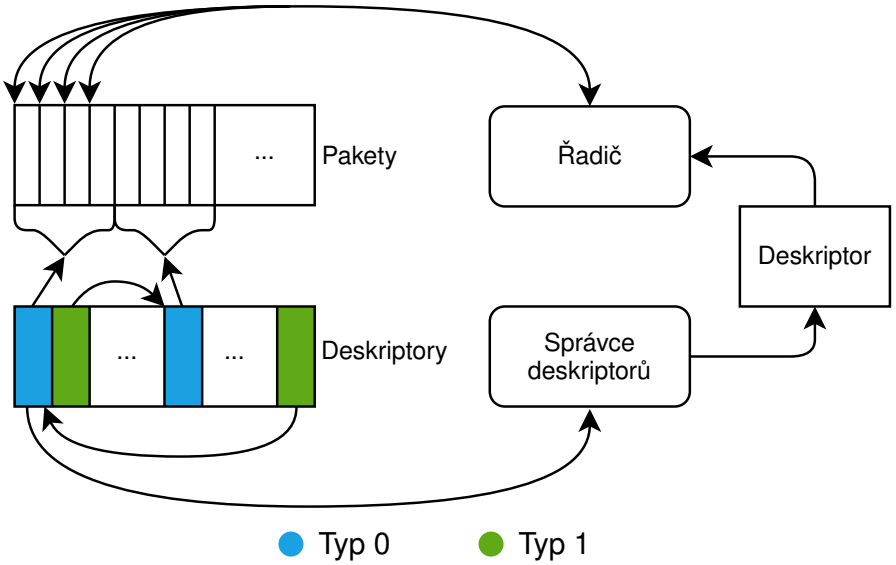
2.3 Network Developement Kit

Network Developement Kit (NDK) je prostředí vyvíjené oddílem Liberouter v rámci sdružení CESNET sloužící k vytváření nových síťových aplikací založených na FPGA (Field-programmable gate array) akceleračních kartách. Obsahuje např. síťové moduly založené na standardních ethernetových Hard IP, DMA moduly postavené na PCIe nebo automatizované skripty pro syntézu celých designů. V této práci budou podrobně popsány již zmíněné DMA moduly FPGA akceleračních karet. Následující kapitola čerpala ze zdroje [15].

DMA řadiče v rámci této platformy využívají mimo řídící registry i speciální metadata nazývané deskriptory a hlavičky. Deskriptory popisují vyrovnávací paměť, kterou DMA modul využívá pro přenášená data. Přenášená data jsou dělena do paketů, protože tyto DMA slouží k síťovému přenosu. Jednotlivé pakety pak popisují hlavičky, které jsou paketu v paměti vždy předřazeny.

2.3.1 SZE

DMA modul SZE využívá univerzální návrh kompatibilní s většinou karet podporovaných v rámci NDK platformy. Funguje na principu proudových přenosů, tj. shlukuje pakety v paměti do jednoho proudu. To zjednodušeně znamená, že se pakety zpracovávají sekvenčně. Následný obrázek popisuje vnitřní strukturu modulu.



Obrázek 2.1: Pakety a deskriptory jsou umístěny v datových bufferech (vyrovnávací paměť). Při přenosu řadič získává deskriptory od správce deskriptorů a ty pak používá k adresaci. Řadič takto obdrží začátek adresy s daty a délku deskriptorem odkazovaného bloku. V této fázi může sekvenčně číst, nebo zapisovat. Před každým paketem je jeho hlavička, která oznamuje, kde paket začíná a jakou má délku. Hlavičku pro příchozí pakety musí vyrobit řadič a pro odchozí pakety aplikace. Když řadič zná délky paketů, inkrementuje adresu přečtenou z deskriptoru za každý zpracovaný paket, přitom si ale musí hlídat, aby odkazovanou oblast nepřekročil. Pokud by měl přistupovat na místo, které již není popisované tímto deskriptorem, vyžádá si nový. Zároveň může deskriptor odkazovat i na jiný deskriptor (na obrázku druhý deskriptor v pořadí), jak bude uvedeno dále.

Deskriptory jsou při inicializaci vytvořeny ovladačem a během běhu jsou pak pouze používány. Původně byl pro každou stránku paměťové oblasti určené paketům vytvořen deskriptor. V pozdějších iteracích návrhu může deskriptor zastřešovat hned několik stránek. Aby mohl řadič stále vyčítat deskriptorový buffer cyklicky, musel být přidán mechanismus, kterým bude řadič informován o tom, že čte poslední deskriptor. Tato komplikace byla eliminována vytvořením dvou druhů deskriptoru:

Bity	Popis
63 – 12	Fyzická adresa bez spodních 12 bitů (oblast musí být stránkově zarovnaná).
11 – 1	Velikost kontinuální paměti, vyjádřena počtem stránek.
0	Hodnota vyjadřující typ deskriptoru. V tomto případě 0.

Tabulka 2.1: SZE deskriptor typu 0 (direct)

Bity	Popis
63 – 1	Fyzická adresa dalšího bloku s dekriptory.
0	Hodnota vyjadřující typ deskriptoru. V tomto případě 1.

Tabulka 2.2: SZE deskriptor typu 1 (pointer)

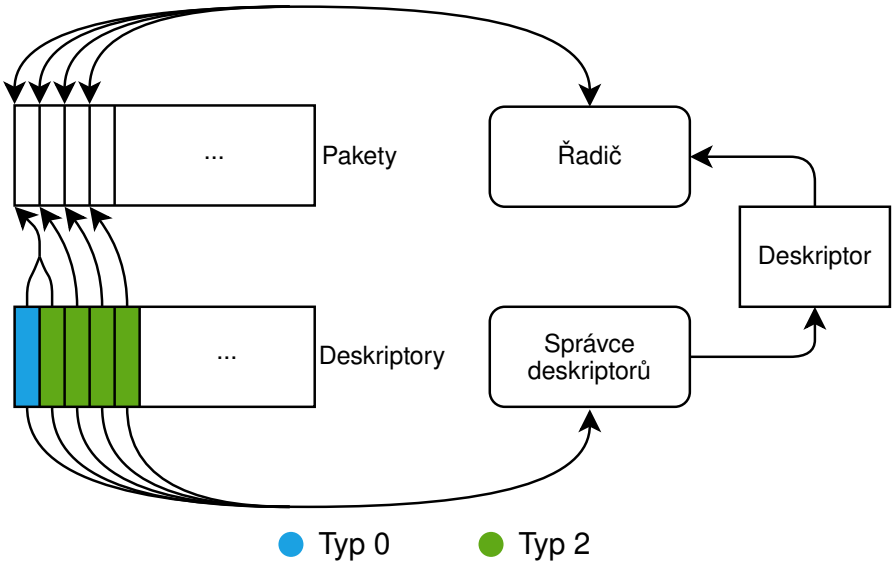
Díky tomu, že SZE přenosy využívají pro data celou stránku, tedy od začátku, deskriptory mohou adresovat oblasti, které jsou stránkově zarovnané. Proto lze u deskriptoru typu 0 nahradit 12 bitů adresujících přesné místo ve stránce jinými metadaty. Další výhodou použití těchto dvou typů deskriptoru je, že oblast pro deskriptory nemusí být zcela kontinuální. Deskriptor typu 1 slouží k odkazování na jiný deskriptor. Může ukazovat zpět na první deskriptor v poli (tím zajistí, že se bude buffer moct číst cyklicky), nebo na následující deskriptor, který pouze nenavazuje fyzickou adresou. Takto tedy jedním deskriptorem typu 0 mohou být adresovány až 4MB ($2^{10} \cdot 4096$) dat. Na začátku každé oblasti je vždy hlavička, která v sobě obsahuje délku paketu pro který slouží. Taková hlavička je předřazena každému paketu.

Pro synchronizaci user space a kernel space (potažmo software a hardware) se používají softwarové a hardwarové ukazatele. Oba ukazatele pracují jako přetékající čítač. Jejich funkce se lehce liší podle směru toku dat. Tyto směry jsou označovány jako Rx (Received – přijímací strana) a Tx (Transmited – odesílací strana). V Tx směru se nejdříve inkrementuje softwarový ukazatel (swptr) o velikost rámce (rámec = paket + hlavička) pro každý paket zapsaný k přenosu a poté hardwarový ukazatel (hwptr) za každý již přenesený paket. V Rx směru je to přesně naopak, prvně je zvýšen hardwarový ukazatel za každý paket přenesený z síťové karty do operační paměti, a pak softwarový ukazatel pro všechny pakety přijaté aplikací.

2.3.2 Medusa

Tento modul byl specificky navrhnut pro síťovou kartu řady COMBO z oddělení Liberouter. Jedná se o kartu COMBO-200G2QL, která dosahuje propustnosti až 200 Gb/s. Hlavní rozdíl tohoto a předchozího modulu spočívá ve způsobu přenosu a to tak, že místo proudového přenosu implementuje přenos paketový. Pakety jsou přenášeny jednotlivě a nezávisle na sobě. To je zaručeno deskriptory.

Deskriptory, tak jako u SZE, jsou 64bitové, aby byly lehce přístupné pomocí indexu (přičtením pořadí deskriptoru k adrese začátku vyrovnávací paměti). Mohou být několika typů, ale pro základní funkcionalitu je zapotřebí právě dvou. Jsou to typ 0 a typ 2. Následující tabulky 2.3 a 2.4 popisují sémantiku jednotlivých bitů deskriptorů.



Obrázek 2.2: Pakety a deskriptory jsou umístěny v datových bufferech. Řadič od deskriptor manageru přebírá deskriptory, které využívá k adresaci. K adresaci jednoho paketu mohou být použity až dva deskriptory, tedy adresa může být rozdělena na dvě části. Řadič si pamatuje poslední vrchní část adresy, kterou obdržel od deskriptor manageru. Pro získání úplné adresy musí požádat o další deskriptor, ve kterém se nachází zbytek (tedy spodní část) adresy paketu. Vrchní část adresy často bývá pro více paketů stejná a proto se používá do té doby, než se obdrží nová. Konkrétně na obrázku první dva deskriptory označují jeden paket, s tím, že první deskriptor obsahuje vrchní část adresy a druhý pak spodní. Následné deskriptory pak už odkazují jednotlivé pakety, protože vrchní část je doplňovaná adresou ze zapamatovaného prvního deskriptoru.

Bity	**Popis**
63 – 62	Hodnota vyjadřující typ deskriptoru. V tomto případě 0.
61 – 35	Rezervováno (v současném stavu nevyužito).
34 – 0	Vrchní část fyzické adresy.

Tabulka 2.3: Medusa deskriptor typu 0

Bity	Popis
63 – 62	Hodnota vyjadřující typ deskriptoru. V tomto případě 2.
61 – 59	Bity pro speciální operace.
58 – 48	Rezervováno (v současném stavu nevyužito).
47 – 32	Délka bloku.
31	Bit pro vygenerování přerušení (nemusí být podporováno).
30	Rezervováno (v současném stavu nevyužito).
29 – 0	Spodní část fyzické adresy.

Tabulka 2.4: Medusa deskriptor typu 2

Protože pracujeme se 64bitovým systémem, může mít fyzická adresa až 64 bitů. Proto jestli má 64bitový deskriptor nést i nějaká jiná data, je potřeba rozdělit adresu na dvě části. K tomuto účelu přímo slouží deskriptor typu 0, který je ve své podstatě pouze pomocný a neodkazuje na žádné konkrétní data. Když jej přečte řadič, uloží si hodnotu vrchní části adresy a pak ji připojí ke každé spodní části adresy, jenž nesou deskriptory typu 2. Tímto postupem získává řadič fyzické adresy do té doby, než přečte další deskriptor typu 0. V tu chvíli si opět uloží vrchní adresu a analogicky pokračuje. Čím víc bude vyrovnávací paměť fragmentovaná (virtuální paměť bude tvořena více fyzickými bloky, které nebudou kontinuální), tím víc deskriptorů bude typu 0.

Pro synchronizaci jsou důležité ukazatele (čítače). Slouží jako komunikační prostředek mezi aplikací a řadičem. V případě odesílání informují řadič, že jsou připraveny nové pakety k odeslání. V případě příjímání naopak informují aplikaci, že byly nové pakety přijaty. Tento modul konkrétně implementuje tyto ukazatele:

- `shp` – software header pointer (softwarový hlavičkový ukazatel)
- `sdp` – software descriptor pointer (softwarový deskriptorový ukazatel)
- `hhp` – hardware header pointer (hardwarový hlavičkový ukazatel)
- `hdp` – hardware descriptor pointer (hardwarový deskriptorový ukazatel)

Výsledek odečtení softwarových a hardwarových hlavičkových ukazatelů lze do jisté míry chápat jako počet paketů ke zpracování. To stejné lze říci o deskriptorových ukazatelích.

Hardwarové ukazatele jsou řadičem obsluhovány interně a popisují stav kanálů. Ovladač k nim má přístup pouze pro čtení. Naopak softwarové ukazatele jsou řízeny ovladačem. Ukazatele jako takové jsou čítače, které jsou inkrementovány pro každý zpracovaný paket nebo deskriptor. Mají maximální hodnotu, které mohou dosahovat, což souvisí s konečností paměťového bloku určeného pro deskriptory a hlavičky. Při dosažení maximální hodnoty ukazatele nastane vynulování, které se dá chápat jako přetečení čítače. Tímto způsobem jsou buffery využívány kruhově. To znamená, jakmile bylo využito všechno místo určené např. pro deskriptory, začne se buffer přepisovat od začátku.

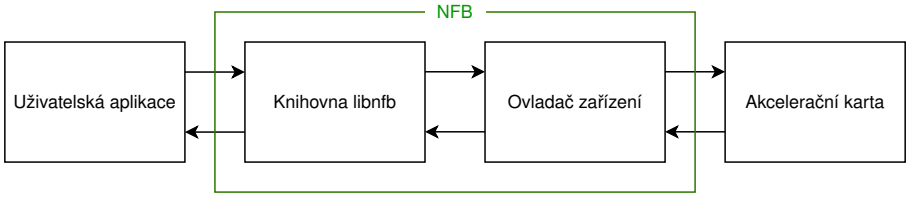
Při odesílání prvně ovladač pro každý paket vytvoří potřebný počet deskriptorů (tedy jeden nebo dva). Inkrementuje softwarový ukazatel, který je zapisován do registru řadiče. Je důležité zdůraznit, že při odesílání se používají jenom deskriptorové ukazatele, protože

komunikace je pouze jednostranná (software zapisuje do firmwaru). Navíc lze z deskriptorových ukazatelů dopočítat, o kolik paketů se jedná. Nejjednodušší je spočítání deskriptorů typu 2. Jakmile řadič zjistí, že hardwarový ukazatel je v kruhovém čítači pozadu, začne číst deskriptory, přenášet jednotlivé pakety a inkrementovat hardwarový ukazatel, než se se softwarovým vyrovnají.

Při příjmu dat je důležitá inicializační část. Při té jsou vytvořeny nejdříve deskriptory prázdných míst v paměti a `sdp` odpovídající jejich počtu je zapsán do registru. Po příchodu dat na rozhraní jsou pomocí DMA přesunuty na místa již popisovaná deskriptory a je inkrementován `hdp` o počet využitých deskriptorů. Zároveň s tím je zvýšen i `hhp` o počet přijatých paketů, který řadič získá od rozhraní. Softwarová část pak pomocí ovladače zjistí počet nově zaplněných deskriptorů tím, že odečte `sdp` od `hdp` a zohlední potenciální přetečení pomocí operace modulo (Pro buffery velikosti 2^n stačí vymaskovat hodnotou $2^n - 1$). Na toto množství deskriptorů pak nahlédne a dívá se na typ deskriptoru. Pouze za deskriptor typu 2 inkrementuje `shp`. Zároveň si udržuje počet nově uvolněných deskriptorů. Jestliže tato hodnota přesáhne určitou mez, jsou dovytvořeny nové deskriptory.

2.4 Network FPGA Board

Network FPGA Board (NFB) je označení pro soubor softwarových nástrojů, knihoven a ovladačů používaných s NDK. Zahrnuje i konfiguraci komponent jako například MI32 nebo řízení datových přenosů (pomocí DMA). Následující kapitola čerpala ze zdrojů [15] a [11].



Obrázek 2.3: Jedním z hlavních účelů této platformy je vytvořit softwarové application programming interface (API) nejen pro moduly DMA. To tedy tvoří knihovna libnfb (dále označováno jako knihovna) a ovladač zařízení.

Pro použití NFB k přenosu dat je nutné nejdříve nastavit prostředí. Knihovnu jako `.so` soubor lze jednoduše přilinkovat při překladu aplikace. Ovladač je zapotřebí vložit do kernelu ve formě modulu. Při vložení NFB modulu se volá inicializační funkce, která vytvoří potřebné datové struktury, alokuje nezbytné datové úseky v paměti (např. pro práci DMA) a také vloží do stromu zařízení (o něm dále v textu) pomocná metadata. Odstranění modulu z kernelu dealokuje paměťové úseky a obecně uvolní používané zdroje.

Před jakoukoliv prací s kartou NFB vytvoří ovladač znakové zařízení, které této síťové kartě odpovídá. Zároveň s tím je provedena konfigurace FPGA v rámci akcelerační karty. Toto zařízení je dostupné v cestě `/dev/nfb[id]`, kde `[id]` označuje identifikátor jednotlivých síťových karet, jež mohou být dostupné ve stejnou chvíli. Toto znakové zařízení je poměrně nestandardní, protože jeho primární systémové volání jsou `ioctl` (I/O control – vstupně–výstupní řízení) a `mmap`. Nicméně `open` a `close` jsou využity klasicky, `write` použito není a `read` je využito následně.

Čtením zařízení je získán tzv. Device Tree (strom zařízení síťové karty) v binární podobě. Strom je tvořen popisem jednotlivých firmwarových komponent karty. Jednotlivé uzly nemají omezen počet potomků a obsahují metadata komponent, které reprezentují. Tato

metadata slouží primárně pro komunikaci knihovny a ovladače, nejčastěji směrem z ovladače do knihovny. Mohou to být například speciální identifikátory pro systémové volání `mmap`, nazývané offsety. Tento strom je pak zpracováván knihovnou `libfdt`, která v něm vyhledává a získává z něj informace ohledně organizace paměťového prostoru zařízení.

Komunikace směrem z aplikace do ovladače probíhá přes systémové volání `ioctl`. Pomocí něho se spouští, v průběhu synchronizuje a nakonec zastavuje přenos. Volání v kernel módu přebírá ovladač zařízení a dekódováním předaného identifikátoru operace, se větví jeho chování. Toto volání podporuje i předávání dat, ale pro jednoduchost jsou takto předávaná data velmi malá a obsahují pouze informace o přenosu (např. ukazatele, identifikátory kanálu a jiné).

Část ovladače pro přenos dat skrz DMA byla navrhnuta tak, aby mohl sloužit více uživatelským aplikacím i jeden kanál. Zajišťuje to systém přihlašování ke kanálům. Když požaduje aplikace po kanálu přenášení dat, musí se vytvořit interní datová struktura, která ji označuje. Nazývá se odběratel, nebo subscriber. Takový odběratel pak může mít různé požadavky na různé kanály. Každý požadavek musí být nejdříve zaregistrován. Toto je označováno jako odběr, nebo subscription. Subscription se už váže na konkrétní DMA kanál. Odběratel může mít obecně neomezený počet odběrů na libovolných kanálech, nehledě na to, zda se jedná o odchozí nebo příchozí stranu komunikace. Jednotlivé odběry využívající stejných kanálů se pak musejí synchronizovat. Když aplikace požaduje pracovat s pakety, musí o ně nejdříve ovladač požádat neboli uzamknout. Ovladač zkontroluje, zda může počet potřebných paketů aplikaci poskytnout. Pokud ne, nabídne maximální dostupné množství.

Již zmíněná knihovna libnfb deleguje požadavky od aplikace na ovladač. Vytváří pro aplikaci jednotné rozhraní pro různé konfigurace FPA, ale zároveň komunikace s ovladačem probíhá ve stejném stylu. Nehledě na firmwaru musí při jejím použití, pro přenos síťových dat, aplikace dodržet základní strukturu požadavků (v tomto pořadí):

1. Otevření znakového zařízení.
2. Otevření potřebných DMA kanálů a přihlášení k nim.
3. Spuštění potřebných DMA kanálů.
4. Získání paketového burstu.
5. Navrácení paketového burstu.
6. Zastavení DMA kanálů.
7. Zavření DMA kanálů.
8. Zavření znakového zařízení.

Krokem 1 se zároveň i vytvoří datová struktura `dev`, která obsahuje všechny důležité metadata pro další operace. Kroky 2 až 7 používají systémové volání `ioctl` pro komunikaci s modulem DMA pomocí ovladače. Krok 2 zahrnuje vyhledávání ve stromě zařízení pomocí knihovny `libfdt`. Podle verze DMA modulu se vyhledávají různé komponenty. Ze stromu pak lze vyčíst potřebné offsety, které jsou uloženy. Zároveň jsou pro tento účel vyčteny i délky těchto úseků. Následně se pomocí `ioctl` přihlašuje k požadovanému DMA kanálu. Ve 3. kroku se spouští jednotlivé kanály opět pomocí `ioctl`. Ovladač kontroluje zda kanál již není spuštěn. Když je, neprovede se žádná operace. Kroky 4. a 5. spolu úzce souvisí, s tím, že po 5. kroku se opět může skočit na krok 4, a to v podstatě neomezeně krát. Lze přenést

i více paketů na jedno volání funkce. Této skupině přenášených paketů se pak říká burst. Požadavek na přenos dat znamená v Tx a Rx směru lehce odlišné věci. V obou případech je knihovně kromě počtu paketů potřeba předat ještě identifikátor již otevřeného kanálu (handle) a paměťový buffer. Paměťový buffer je v tomto případě pole, ve kterém jsou jednotlivé pakety reprezentovány datovými strukturami, které obsahují např. adresu dat paketu. Pro Tx i Rx jsou adresy datových struktur v tomto bodě naplněny. Pro Tx tyto adresy znamenají místa, kam může aplikace data k odeslání zapsat a pro Rx tyto adresy znamenají místa, ze kterých může aplikace data číst. Zároveň zde může knihovna komunikovat s ovladačem, ale pouze když je to nutné (nelze poskytnout požadovaný počet paketů, a proto je potřeba provést synchronizaci s ovladačem). 5. krok je důležitý pro synchronizaci s ovladačem a potažmo i řadičem. Slouží k tomu, aby se mohly srovnat ukazatele knihovny, ovladače i řadiče. Pro odchozí stranu slouží i k tomu, aby se DMA kanál dozvěděl, že aplikace připravila pakety k odeslání. Pro příchozí stranu zase slouží k tomu, aby se DMA kanál dozvěděl, že aplikace přijaté pakety již zpracovala. Po ukončení přenosu následuje 6. krok. V rámci něho se ovladači oznamuje odhlášení od konkrétního kanálu a zároveň jeho zastavení v případě, že už k němu není přihlášen nikdo jiný. Předposlední, tedy 7. krok, slouží k uvolnění struktur alokovaných pro řízení komponent síťové karty z operační paměti. 8. a poslední krok zaštiťuje konečné uvolnění struktury reprezentující NIC a nejčastěji i ukončení aplikace.

2.5 Principy přenosu síťových dat

Když vznikala potřeba přenosu a sdílení dat mezi koncovými stanicemi, bylo nutné navrhnout nějaký univerzální způsob pro komunikaci počítače s vnějším okolím. Tím se stal Ethernet vyvíjený firmou PARC (tehdy Xerox PARC). Protože CPU nemůže zpracovávat data síťovou rychlostí, bylo třeba navrhnout komponentu, která mu od této zátěže odlehčí. Tím se stala síťová karta, která je v dnešní době běžně přítomná na základní desce osobních počítačů. Pro síťové uzly ale tato karta není dostačující. Proto se používají NIC, nejčastěji připojované na PCI sběrnici. Vestavěné síťové karty obecně nejsou využívány z důvodů, že nedosahují nejnižších latencí, protože tento požadavek běžný uživatel nemá. Obsluha síťových karet může být poměrně různorodá. S neustále narůstající poptávkou po vyšších propustnostech je snaha vyvíjet nové technologie a platformy, které tento přenos urychlí. Některé jejich principy budou nadále popsány, počínaje základním modelem užívaným drtivou většinou osobních počítačů [12].

2.5.1 Standardní model

V klasickém pojetí přenosu síťových dat je hlavní součástí síťové rozhraní. Je do jisté míry podobné znakovému nebo blokovému zařízení, ale není umístěno v souborovém systému a nepoužívá operace `read` a `write`. V systému jich obecně může být mnoho a všechny jsou ovládány síťovým ovladačem. Jako jsou všechny ovladače je i tento modul. Při jeho zavedení do systému jsou pro všechna rozhraní vytvořeny datové struktury jim odpovídající.Jejich inicializace probíhá při nahrání modulu, což může být při bootování (zavádění) kernelu, ale i později při manuálním načtení. Další nezbytný krok je otevření rozhraní. To obsahuje dvě systémová volání, která slouží k přiřazení adresy rozhraní a jeho spuštění. V rámci toho je i spuštěna odchozí fronta. Dále už následuje samotné přenášení dat [2].

Linuxový síťový provoz obecně využívá TCP/IP stack implementovaný v kernelu. Při odesílání dat obaluje paket potřebnými hlavičkami a při přijmu zase data paketu z hlaviček vyjímá. Kernel pro user space abstrahuje práci s rozhraním pomocí tzv. socketů. Je k nim

přistupováno pomocí souborových deskriptorů a jsou reprezentovány datovou strukturou. Každý síťový paket patří nějakému socketu a v určité fázi přenosu se nachází v paměťovém bloku jeho datové struktury. Při odesílání je pouze zavolána funkce, která zkontroluje paket a předá jej funkci specifické pro daný hardware. Pakety jsou odesílány jednotlivě. Přijímání dat je o něco složitější. Dá se provést pomocí dvou způsobů: pollování (aktivní vyčítání) nebo řízení přerušením. Přerušením řízené přijímání dat ale bývá běžnější. Přerušení je použito jak pro příjem, tak pro od odesílání. Při odesílání je přerušení vyvoláno ve chvíli, kdy byl paket odeslán a při příjmu když paket dorazil. Obsluha přerušení pro odesílání je ale prostější, a to v tom, že se typicky jen aktualizují statistiky. Pro příjem je nutné převzít paket z fronty rozhraní, přístupem k jeho datové struktuře. Pro tento paket nadále alokovat prostor v rámci socketu a přesunout do něj přijatý paket. Součástí této operace je také aktualizace statistik. Pro urychlení přenosu je standardně využito DMA, kde je ale před přijetím paketu nutnost alokovat místo v paměti, do kterého je jej DMA schopno přenést. Přijímání dat pomocí pollování je dosaženo pomocí tzv. NAPI (new API). Ne všechna rozhraní mohou tento druh implementovat. Je pro něj potřeba udržovat až několik paketů v paměti (např. přímo na kartě). V tomto případě se pro první paket stejně použije přerušení, ale pro následující je již vypnuté a vyčítají se aktivně [2].

2.5.2 Data Plane Developement Kit

Hlavní podstatou Data Plane Developement Kitu (DPDK) je minimalizovat komunikaci s hardwarem v kernel space. Místo toho je tato funkcionalita prováděna přímo pomocí knihovních funkcí DPDK. Aby mohly být síťové karty obsluhovány přímo z user space, je zapotřebí zapsat jejich sběrnicové číslo (bus number) do souboru `unbind` a tím odebrat kontrolu síťovému ovladači od této karty. Další požadavek je, že porty musí být spravovány pomocí ovladačů `vfio_pci`, `igb_uio`, nebo `uio_pci_generic`. Díky nim je možné obejít kernelové ovladače vyjma těch používaných k inicializaci zařízení a nastavení PCI rozhraní. Veškerá komunikace mezi aplikací a síťovou kartou pak probíhá pomocí PMD (Poll Mode Driver). Důležité pro DPDK je také to, že všechny paměťové bloky jsou vytvořeny před začátkem přenosu a díky tomu není v průběhu zpomalován alokací prostoru pro každý paket, jak je tomu ve standardním Linuxovém síťovém modelu. Pro samotný přenos jsou pak používány kruhové paměťové bloky, do kterých jsou pakety nejdříve umisťovány. Ty jsou aktivně vyčítány (polling). Pakety jsou reprezentovány pomocí datových struktur, které jsou na počátku alokované a jsou akorát zabírány pakety. Po přijmutí paketu je datová struktura navrácena a může být znovu použita [5, 13, 4].

2.5.3 eXpress Data Path

eXpress Data Path (XDP) je primárně určen pro příchozí stranu komunikace, neboli Rx. Pracuje v nejnižší vrstvě Linuxového síťového modelu a jeho hlavním účelem je zpracovávání dat paketů před kernelem. Konkrétně si vytváří vlastní vykonávací prostředí ve formě virtuálního stroje. Na něm běží eBPF kód, který je rozšířením základního Berkley Packet Filter (BPF). Zakládá si na programovatelnosti aplikace, kterou pak předává kernelu pro kontrolu a řízení příchozích dat. Jakmile je přijat paket, projde TCP/IP stackem a ještě předtím, než se na jeho data začne dívat kernel, je předán aplikaci XDP. Ta vyhodnotí jeho další postup tím, že předá kernelu návratovou hodnotu. Podle ní se rozhoduje jak se má s paketem dál zacházet. Pokud má být například přeposlán na Tx nebo zahozen, k aplikaci se vůbec nedostane a tedy i není proveden žádný přesun v paměti [6].

Kapitola 3

Zhodnocení aktuálního stavu

V současné době je základní Linuxový síťový stack pro vysokorychlostní přenos paketů v podstatě nepoužitelný. I když pro standardní síťový provoz funguje dostatečně, jeho fundamentální charakteristiky v zásadě přímo zabraňují nízké latenci. Například alokace místa pro každý paket a následné přesuny paměti, nebo komunikace se síťovým rozhraním přes kernel space je pro určitou hustotu provozu příliš zpožďující. Pochopitelný krok, je tedy tyto operace co nejvíce omezit, nebo úplně vyloučit z modelu.

Jako nadějná alternativa se tedy jeví platforma DPDK, která minimalizuje počet zpomalujících operací v průběhu přenosu. Je vyvíjena v rámci firmy Intel, která je aktuálně na tomto trhu jedním z vedoucích konkurentů. Nicméně v tuto chvíli je projekt open source a přispívá k němu tedy i širší veřejnost. Mimo jiné řeší i již zmíněné vlastnosti, které bylo vhodné eliminovat. Pomocí jednoduchého ovladače dokáže s kartou komunikovat napřímo v user space, a tím v průběhu přenosu obchází přepínaní kontextu, které je potřebné při interakci user space s kernel space. Problému s dynamickou alokací předchází alokováním určitého místa rozděleného pro pakety při inicializaci a poté recyklováním těchto míst v následném běhu. Zároveň není třeba paket v paměti již nikam přesouvat, protože po příchodu jde na předalokované místo v paměti, které je aplikaci přístupné.

XDP je další způsob, kterým se dá celý proces urychlit. Je sice používáno pouze pro příchozí provoz, ale i přesto může mít na přenos velký vliv. Již zmíněný problém s dynamickou alokací je totiž problémem primárně v Rx směru. Proto je dobré, když nebude nutné pro některé pakety alokovat místo. Sníží se tím tedy zátěž. Nicméně při srovnání největších výkonností dosažených pomocí každé z platforem vítězí DPDK.

Pro tuto práci je ale klíčová platforma NFB, která zajišťuje manipulaci s daty pro karty COMBO vyvíjené ve sdružení Cesnet. Jejich přístup je v jistých ohledech inspirovaný různými platformami. Například podobnost se standardním modelem je ve využívání komunikace s hardwarem pomocí kernel space. S DPDK je příbuznost ve využívání deskriptorů pro jednotlivé pakety, a s tím tedy i paketové přenosy jako takové. Ty jsou ovšem jen pro jeden druh firmware, kterým je Medusa, jak už bylo řečeno. Zároveň ale i třeba alokace místa pro pakety také probíhá před přenosem, nebo jsou pakety získávány pollováním. S těmito podobnostmi přichází i výhody a nevýhody původních implementací. Tedy některé již zmíněné problémy NFB řeší, a některé ne úplně. Konkrétně operace s pamětí jsou minimalizovány, ale průběžná komunikace s kernelem zůstává. Má to ale svůj důvod. Díky této komunikaci lze jednotlivá rozhraní používat současně více aplikacemi, což může mít v určitých případech jisté výhody.

Mezi dvěma návrhy firmware pro FPGA akcelerační karty jsou značné rozdíly, které také mají velkou roli na latenci. Tato skutečnost se samozřejmě promítá i do softwaru jednotli-

vých modulů. Hlavní odlišnost je v módu přenosu, a to tedy mezi paketovým a proudovým. Jejich latence se odvíjí od míry paralelizace zpracování paketů. Proudové přenosy obecně musejí číst pakety sekvenčně, zato paketové nemusí. Nicméně paketové přenosy mají i nevýhodu. Deskriptory pro pakety musí být vytvářeny za běhu. S tím souvisí i to, že řadič musí v paketovém režimu číst deskriptory pro jednotlivé pakety, kdežto v režimu proudovém je deskriptor čten mnohem méně často, protože deskriptory popisují více paketů.

Kapitola 4

Návrh

4.1 Motivace

Hlavní smysl práce je urychlení přenosu paketů mezi uživatelskou aplikací a síťovým rozhraním. Tedy nalézt tzv. bottleneck (zúžené hrdlo) platformy NFB, navrhnout a implementovat řešení, které celý výsledný přenos urychlí. I když je toto primární požadavek této práce, není jediný. Jedna z hlavních myšlenek NFB je to, že poskytuje uživatelským aplikacím jednotný softwarový přístup k různým kartám. To znamená, že nehledě na firmwarové verzi nebo návrhu karty, by aplikace měla využívat stejné funkce ve stejném pořadí a také by měla dostávat stejnou zpětnou vazbu. S tím souvisí i skutečnost, že jakákoliv změna ve vnitřní struktuře, ať už knihovny nebo přímo ovladače by se neměla promítnout do uživatelské aplikace. Neboli, že změna v platformě nevyžaduje úpravu v aplikaci. A proto je tedy důležité, aby všechny nové verze byly zpětně kompatibilní. To se tudíž vztahuje i na libovolné úpravy v rámci této práce. Jelikož platforma v současnosti podporuje základní dva firmwarové návrhy, je i toto třeba zohlednit a zaručit funkcionalitu pro oba moduly.

4.2 Identifikace problému

Předtím, než lze problém začít řešit, je potřeba prozkoumat obecné principy a interní organizaci jednotlivých modelů a zjistit, které operace a charakteristiky lze optimalizovat, či zcela vyřadit. Na tuto záležitost se dá podívat ze dvou úhlů pohledu, které spolu ale souvisí. Prvním je standardní statická analýza jednotlivých funkcí knihovny, jejich návazností, komunikace a obecných principů, které jsou v platformě využity. Druhým je dynamická analýza ve formě zkoumání spouštěním jednotlivých úseků, kterými data při přenosu putují a jejich současného měření. Už byly zmíněny jisté předpoklady pro nízkolatenční přenosy dat, na které je dobré se v první řadě podívat, než budou zkoumány skryté vnitřní mechanismy.

Nejzákladnějším je způsob příjmu paketů a to, zda je využit režim s generováním přerušení, nebo režim pollující. NFB aktivně kontroluje, zda nepřišly nové pakety a vyhýbá se díky tomu přepínání kontextů a režii s tím spojenou. To znamená, že tento způsob odpovídá pollování, které je časově přívětivější. Dalším z nich je vyloučení alokace paketů v paměti přímo za běhu. To platforma splňuje používáním předalokovaných deskriptorů pro řadič DMA. V modulu Medusa jsou sice deskriptory modifikovány při přenosu dat, což přidává mírnou latenci, ale ta je zanedbatelná, obzvlášť v porovnání s alokací paměti. Poslední potenciální překážkou může být komunikace user space aplikace s kernel space ovladačem. Při inicializaci akcelerační karty jsou samozřejmě operace toho využívající neškodné.

Jestliže chce ale knihovna komunikovat s ovladačem za běhu, musí využít systémových volání, které zahrnují přepínání kontextu. Přestože je přepnutí kontextu mezi user módem a kernel módem časově méně náročná operace, než mezi dvěma user space procesy, může se při opakovaném volání prodražit. Tím lze tedy na první pohled odhadnout, že toto je potenciálně zpomalující prvek přenosu. Aby se to ale dalo říct s jistotou, je nutné prozkoumat platformu hlouběji.

Je důležité nahlédnout na jednotlivé operace prováděné pouze v rámci přenosu, protože spouštění a zastavování provozu nemá vliv na latenci a je tedy zbytečné tyto části prozkoumávat. Na začátek je dobré se podívat na způsob předávání dat v Tx směru.

Když chce aplikace odesílat data, využívá k tomu funkci knihovny, kterou požádá o adresy, na které může zapsat pakety k přenosu. Tyto adresy jsou předalokované a jsou určené k tomu, aby je mohl modul DMA použít k přenosu dat, ať už se jedná o přenos z akcelerační karty do RAM, nebo naopak. Tyto adresy předává knihovna jejich kopírováním do aplikace, což je nežádoucí chování. To znamená, že tuto funkcionalitu by opět bylo výhodné eliminovat. Nicméně v rámci tohoto kopírování je i prováděna synchronizace s kernelem pomocí systémového volání. To obsluhuje ovladač zařízení, který pracuje v kernel módu. Při obdržení synchronizačního volání dělá ovladač řadu věcí:

1. Příjem dat od user space
2. Vyhledání požadované subscription
3. Synchronizace subscription
4. Navrácení dat user space

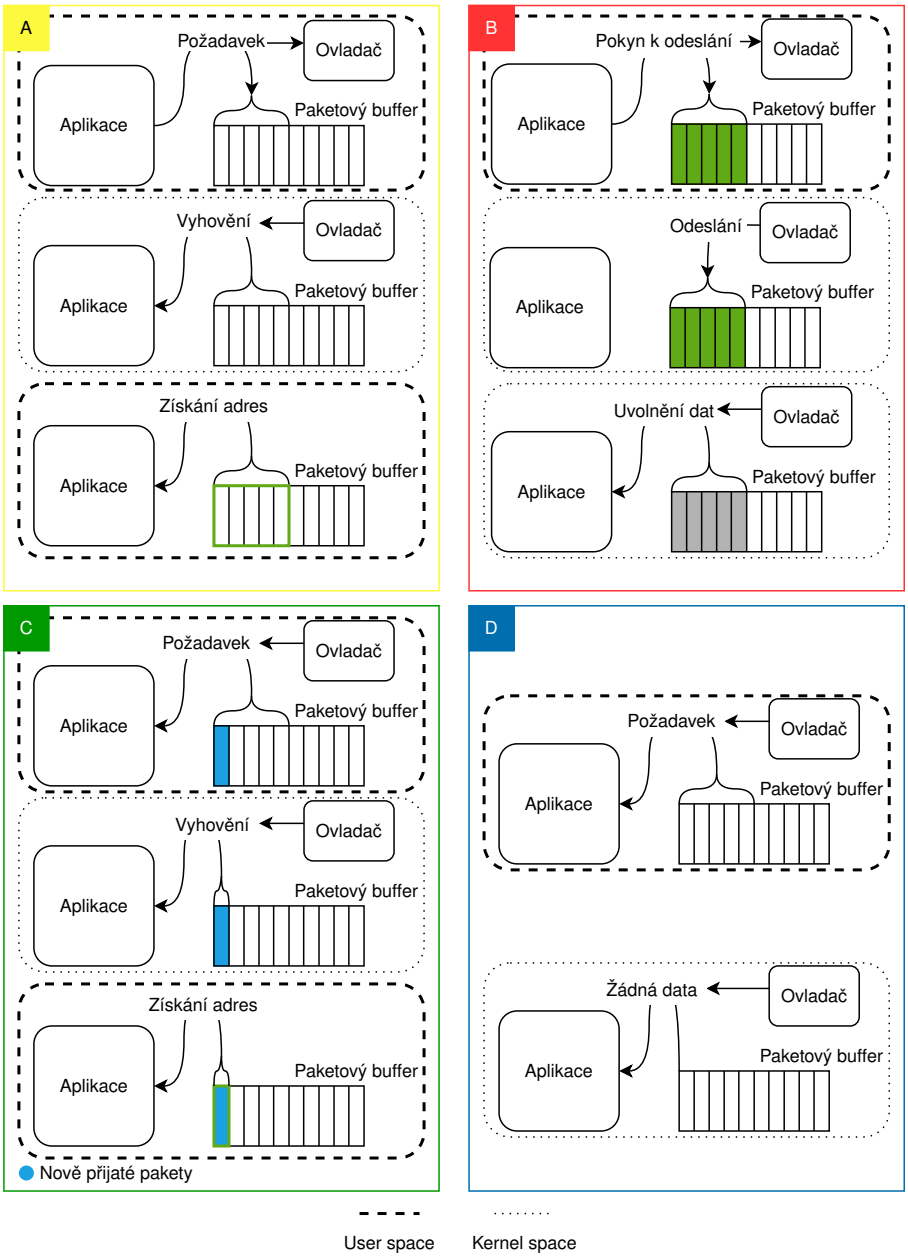
Na první pohled je vidět, že v rámci 1. kroku data putují mezi user space a kernel space tam a zpátky, což se dá považovat za další známku toho, že systémové volání je dobré odstranit. Dále si lze povšimnout, že oba dva další kroky (tj. 2. a 3.) slouží primárně k obsluze systému odebírání kanálů v rámci více procesů. Tento systém byl sice navržen ke svému účelu, ale je nutné se rozhodnout, zda je nezbytný i přes jeho zpomalující vliv na přenos. Přestože 3. krok primárně slouží pro synchronizaci odběratelů, jeho část se stará i o srovnání softwarových a hardwarových ukazatelů. V případě modulu Medusa jsou v rámci toho potřebné i určité operace s deskriptory. Tato část je pro přenos klíčová a nelze ji vynechat.

Pro Rx směr se dají vytknout v podstatě stejné věci jako pro Tx směr. Jediný markantní rozdíl zde je ale skutečnost, že funkce pro získávání přijatých paketů se využitím pollovacího módu, volá nepřetržitě. Je tedy ještě důležitější, aby byla časově co nejkratší v případě, že nedorazí nové pakety. Aby mohla tedy s co největší frekvencí kontrolovat dostupnost paketů.

V tomto bodě, když se ukázalo, že se z návrhového hlediska některé částí jeví latenčně jako problémové, je na čase změřit dobu, kterou stráví proces v jednotlivých úsecích. Je důležité zajistit, aby samotné měření přidávalo minimální latenci k průběhu a tedy, aby se co nejméně promítlo do výsledku. Nicméně jde o měření orientační a důležité jsou především poměry hodnot. Měření je rozděleno pro oba DMA moduly zvlášť. I když jsou oba mírně odlišné, knihovna libnfb s nimi pracuje pomocí podobné, ale ne zcela stejné sady funkcí. Práce knihovny se dá rozdělit na čtyři různé typy:

A – Získávání (připravení) paketu v Tx směru

B – Odesílání paketu v Tx směru



Obrázek 4.1: Grafické znázornění jednotlivých měřených typů.

C – Získávání paketu v Rx směru (s příjmem paketu)

D – Získávání paketu v Rx směru (bez příjmu paketu)

Tyto typy odpovídají akcím prováděným v průběhu zpracovávání paketu (podrobněji na obrázku 4.1). Největší rozdíly se pak objevují až v ovladači při synchronizaci ukazatelů. Způsob zaznamenávání délek úseků bude pomocí ukládání časových značek na různých místech v kódu. Z těchto časových údajů je pak vytvořena doba trvání odečtením předchozí hodnoty od novější. Tyto výpočty jsou prováděny pro deset tisíc běhů (každý běh pracuje s jedním paketem), ze kterých je na závěr vytvořen aritmetický průměr a směrodatná od-

chylka (odchylka je vždy druhá hodnota v závorkách). První měření tedy bude na modulu SZE.

	A	**B**	**C**	**D**
cesta k synchronizaci	740,38 (724,9)	226,14 (218,9)	303,23 (234,55)	66,57 (13,9)
synchronizace	6722,57 (1867,3)	7128,17 (1847,05)	3415,83 (1362,27)	3066,67 (1253,37)
návrat do aplikace	406,41 (297,21)	99,92 (147,83)	3262,5 (1058,76)	115,94 (152,97)

Tabulka 4.1: Latence modulu SZE v nanosekundách

Výsledky v tabulce 4.1 do určité míry odpovídají předpovědi. A to tedy, že největší zpoždění způsobuje synchronizace s kernelem. Je to výrazně nejdelší část celého přenosu a proto by měl být záměr tuto část co nejvíce optimalizovat.

Pro modul Medusa je postup téměř identický. Dá se předpokládat, že synchronizace bude opět zpomalující prvek a v tomto případě ještě výraznější kvůli dynamickým deskriptorům. Protože typy mají stejný funkční význam jako u předchozího modulu, jsou označení ponechána stejná.

	A	**B**	**C**	**D**
cesta k synchronizaci	320,61 (390,33)	211,25 (343,93)	51,68 (364,35)	295,64 (616,6)
synchronizace	4915,89 (2012,9)	3956,45 (2252,69)	1244,16 (1321,23)	1417,14 (1108,46)
návrat do aplikace	509,72 (512,74)	107,38 (334,04)	223,52 (390,49)	191,25 (411,4)

Tabulka 4.2: Latence modulu Medusa v nanosekundách

Z výsledků tabulky 4.2 vyplývá stejná skutečnost. V Tx směru lze vidět nárůst v synchronizaci kvůli deskriptorům, jak bylo předpovězeno. Rx vypadá velmi podobně, protože pro jeden paket se deskriptory ještě nevytváří. Nehledě na to je synchronizace i v tomto typu nejvíce zpomalující prvek.

4.3 Řešení problému

Při snaze o zrychlení DMA přenosu je třeba podívat se na zjištěné nedostatky, které by měly být eliminovány. Je však nezbytné nejprve zvážit, zda je všechny eliminovat lze. Pokud ne, jakým způsobem je jde alespoň upravit, aby při přenosu zabíraly co nejméně času. Pokud ano, jakým způsobem je odstranit, aby tento zásah nenarušil strukturu platformy. Návrh všech prvků je stejně důležitý, protože do výsledné rychlosti se počítají všechny.

4.3.1 Systém odběratelů

Prvním prvkem k úpravě je využívání komunikace user space a kernel space za běhu. Současně je i nejobsáhlejší a zahrnuje některé další části ke změně. Cílem je toto chování úplně vynechat a nezpomalovat jím přenos. Hlavní důvody, proč se v původním návrhu nachází, jsou systém odběratelů kanálů DMA a synchronizace ukazatelů. Odebírání kanálů je nezbytné ve chvíli, kdy je nutné, aby s jedním rozhraním pracovalo více aplikací než je DMA kanálů. V případě této práce to ale není podmínkou. Obecně se tato situace moc často nestane a i proto je důležitější rychlost než počet souběžně přenášejících aplikací. Z toho důvodu lze tuto funkcionalitu zcela vyjmout a ušetřit tím potřebný čas.

4.3.2 Synchronizace

Se synchronizací ukazatelů to tak jednoduché není. Jestliže má knihovna v úmyslu odesílat pakety, musí se o tom nějak dozvědět řadič DMA. Zároveň když knihovna přijímá a kontroluje, zda nejsou dostupné nové pakety, musí se dívat na hardwarové ukazatele řadiče. Tím, že jsou řadič a k němu připojené registry, součástí hardwaru, tak k nim kernel zakazuje uživatelské aplikaci přímý přístup. Knihovna tedy musí využít nějaký jiný způsob. V původní implementaci je přístup řešen pomocí `ioctl`, které za běhu zapříčiňuje právě nežádoucí změnu kontextu. Je tedy důležité se této vlastnosti v novém návrhu vyvarovat. Díky struktuře platformy NFB a tomu, že akcelerační kartu v systému reprezentuje znakové zařízení, obsahující Device Tree, je možný i další typ přístupu. Je to systémové volání, které opět zapříčiňuje změnu kontextu. Rozdíl je ale v tom, že změna probíhá pouze jednou, a to při inicializaci a ne při samotném přenosu. Jedná se o `mmap`. Při čtení zařízení označujícího kartu je získán Device Tree. V něm se dá navigovat pomocí knihovny libfdt a nalézt konkrétní uzel představující reálnou komponentu. V uzlu jsou primárně metadata a offsety komponent v rámci zařízení. Tím pádem lze ze stromu získat i offsety registrů DMA modulu. Při inicializaci lze tyto offsety kanálu použít pro mapování adres pouze hardwarových ukazatelů přímo do například datové struktury v user space. Všechny přístupy k těmto ukazatelům pak může knihovna provádět rovnou z user space bez přepnutí kontextu.

Pro softwarové ukazatele to ale tímto způsobem udělat nelze. Popsaný přístup je aplikovatelný na paměť, kterou knihovna potřebuje pouze číst. Softwarové ukazatele slouží k tomu, aby informovaly DMA modul o stavu přenosu. Důležité je, že pochází, jak už jméno napovídá, ze softwaru a putují směrem do hardwaru. Tedy knihovna potřebuje tyto ukazatele zapsat do karty. To lze ale také vyřešit bez přepínání kontextu. Opět lze využít libfdt pro vyhledání komponenty, přesněji konkrétní Rx nebo Tx DMA fronty a pomocí funkce knihovny libnfb do ní zapsat požadovanou hodnotu ukazatele. Tím jsou tedy vyřešeny oba druhy ukazatelů. Knihovna může jak číst hardwarové ukazatele díky mapování, tak může i zapisovat pomocí funkce k tomu určené, kterou poskytuje knihovna.

4.3.3 Deskriptory a manipulace s pamětí

Co zbývá, je ještě práce s deskriptory. V těch se jednotlivé firmwary liší nejznatelněji a je potřeba si je také prohlédnout. Deskriptory jsou tak jako ukazatele způsob, kterým se dorozumívá hardware se softwarem. Přesněji řečeno software předává informace o alokované paměti, kterou má při své činnosti DMA používat. Ať už pro získávání dat, nebo pro zapisování. Logicky z toho vyplývá, že software musí strávit nějaký čas při vytváření těchto deskriptorů. Oba druhy DMA vytvářejí deskriptory při inicializaci. To je v pořádku, protože připravování kanálu se provádí před samotným během a tím pádem není třeba tuto část

optimalizovat. Díky tomu že modul SZE používá pořád stejné deskriptory, práce pro něj v tomto bodě končí. Na začátku paměť pro pakety rozdělí na kontinuální bloky, které může popsat deskriptory a rozlišování jednotlivých paketů pak nechává na řadiči. Během přenosu už nemusí software nic řešit a všechnu práci odvádí firmware. Proto, když se jedná o implementaci SZE, není třeba převádět žádnou další funkcionalitu do user space.

Medusa na druhou stranu údržbu za běhu potřebuje. Základem tohoto návrhu je paketový přenos DMA. Tedy, že každý paket je zpracováván jednotlivě. Do jisté míry se funkcionalita z hardware přenáší do software. Ovladač musí tentokrát pro každý paket vytvářet deskriptor. Stejně jako v SZE je toto děláno na začátku přenosu. Tady ale nastává problém. Hardware používá 64bitové deskriptory pro oba návrhy firmwaru. Deskriptor musí obsahovat fyzickou adresu, kam se má paket přenést, a zároveň i jeho délku. Obvykle jsou požadavky na deskriptory větší a je od nich očekáváno, že budou přenášet více metadat. Minimálně ale musí přenést tyto základní informace. V tuto chvíli je standardní, že systém, na kterém software poběží bude 64bitový a tedy fyzická adresa může mít i 64 bitů. Kvůli tomu se obě tyto informace nemohou vejít do jednoho deskriptoru. Jsou tato data proto rozdělena do dvou deskriptorů (viz tabulky 2.3 a 2.4). Aby výsledných deskriptorů bylo co nejméně, musí být popisovaná oblast kontinuální. I v tomto případě by ale muselo být deskriptorů více, než paketů. Čím víc by paketový buffer byl fragmentovaný, tím více by muselo být deskriptorů. Zároveň v každém běhu aplikace může být tento buffer různě fragmentovaný a tím pádem by mohl být i různý počet deskriptorů. Proto je tedy nutné zachovat vytváření deskriptorů dynamicky a zachování pevného počtu deskriptorů, které jsou dostupné v jednu chvíli.

Bodem zadání je, že všechny aplikace využívající tuto knihovnu musí zůstat beze změny. To znamená, že kopírování adres mezi aplikací a knihovnou musí být zachováno. Co ale lze udělat, je přidat knihovně podporu i pro nový druh aplikace. Tento druh se kopírování vyhne. Je to aplikace využívající platformu DPDK. DPDK má přímo knihovnu, která tuto funkcionalitu obstarává. Dokáže alokovat prostor v paměti a poté získat i I/O adresu pro jednotlivé pakety, která pak může být použita pro vytváření deskriptorů. Zbývá ale pořád jak tyto adresy předat knihovně. Z rychlostního hlediska se zdá nejvýhodnější způsob, zjištění adres všech paketů a uložení těchto hodnot do pole. To znamená, že v poli bude tolik adres, jako je alokovaných míst pro pakety. Knihovna libnfb pak může k těmto adresám přistupovat bez přidané latence a vytvářet deskriptory za běhu. Je to sice na úkor paměti, protože je potřeba pořád udržovat celé pole, ale zjednodušuje to práci a odstraňuje veškerou režii při přenosu.

S novou DPDK aplikací v mysli byla navrhnuta i změna knihovny pro původní aplikace. V tomto případě, je ale problém, že user space nemá přístup k adresám (tentokrát fyzickým) bufferu pro pakety ovladače. Musí je tedy od kernel space nějakým způsobem získat. Je opět důležité aby toto zjišťování neprobíhalo průběžně při přenosu. Proto se využije stejný způsob jako při použití aplikace DPDK. Fyzické adresy budou knihovně nepřímo předány jako pole. Přesněji řečeno, knihovna si musí při startu DMA kanálu toto pole musí získat od ovladače. Opět se zde nabízí použití stromu zařízení pro uschování offsetu pro tento ukazatel. Knihovna si tedy namapuje pole do své datové struktury, kterou nadále využívá. Z něj pak analogicky čte adresy a průběžně vytváří deskriptory. Ovladač ale předtím musí pole naplnit. To zajistí při inicializaci. Kernel může totiž jednoduše zjistit fyzickou adresy všech míst pro pakety.

Z podstaty platformy NDK a NFB bylo jasné, že nový návrh bude muset být také rozdělený na dvě části. Pro každý modul jedna. Většina součástí knihovny zůstává stejná. Některé části jsou ale například přesunuty do user space a tím je potřeba přidat režii. Ve vý-

sledném návrhu lze tedy celou synchronizaci hardware se softwarem ponechat v user space a tím se vyhnout přepínání kontextu. Dynamické generování deskriptorů je tady zanecháno, díky čemuž může nakonec knihovnu libnfb využít i aplikace DPDK. Kopírování adres je potřeba zachovat kvůli zpětné kompatibilitě, lze jej ale obejít novou aplikací. Výsledná implementace se bude primárně zaměřovat na inicializační proces, ve kterém je snaha provést co nejvíce operací, aby je nebylo nutné provádět za běhu.

Kapitola 5

Implementace

V této kapitole bude podrobně popsána implementace nového návrhu. Základní principy byly již vysvětleny a proto zde budou popsány změny v software, kterými bylo nově navržených funkcionalit docíleno. Také budou vysvětleny části, které byly v návrhu pouze zmíněny, nebo úplně vynechány. Vzhledem k tomu, že je s oběma současnými implementacemi DMA, v rámci knihovny i ovladače, nejčastěji zacházeno pomocí různých funkcí, rozhodl jsem se je i v popisu rozdělit. Knihovna se téměř hned na začátku inicializace podívá do stromu zařízení se kterým modulem pracuje a podle toho se práce větví. Protože návrh s proudovým přenosem DMA je méně komplikovaný, začnu s ním a v rámci něj popíšu i některé univerzální vlastnosti obou druhů.

5.1 SZE

Aplikace volá funkce knihovny ve stejném pořadí, jak bylo popsáno v kapitole 2.4, což znamená, že nejdříve je nutné otevřít znakové zařízení a hned poté lze otevřít požadovaný DMA kanál. První akcí byla původně komunikace s kernelem a typicky požadavek na přihlášení. V novém návrhu se ale systém přihlašování nevyužívá. Proto na tomto místě ovladač kontroluje, zda není kanál již využíván jinou aplikací. Pokud je, navrátí user space chybovou hodnotu a nepovolí mu kanál otevřít. V opačném případě se postupuje stejně jako v předchozím návrhu. Následuje vyhledávání offsetu pro hardwarový ukazatel. Offset je v této situaci využit speciálním způsobem. Nemá v tomto případě význam jako nějaké odsazení (jak by z názvu mohlo vyplývat), ale jako identifikátor. Tento identifikátor se používá pro mapování adresy vyhrazené pro hardwarový ukazatel v ovladači do user space pomocí volání `mmap`. Když tedy ovladač obdrží od knihovny požadavek na mapování s tímto konkrétním offsetem, ví , že se jedná o požadavek na hardwarový ukazatel a namapuje jej do user space. Ví to pomocí registrace `mmap`, kterou je potřeba při inicializaci modulu provést. Zároveň s registrací se i tento offset vytváří a zapisuje se do stromu zařízení. Ukazatele knihovna po mapování uschovává v datové struktuře, která představuje DMA kanál.

Následuje spuštění kanálu. Na tomto místě se nejdříve musí otevřít kanál jako komponenta pomocí speciální knihovní funkce. Kanál je potom reprezentován datovou strukturou, která je nutná pro zápis softwarového ukazatele do hardware. Komponenta je opět vyhledána pomocí knihovny libfdt a pomocí tzv. compatible řetězce (unikátní znakový řetězec reprezentující jednu konkrétní komponentu). V tomto případě slouží jako řetězcový identifikátor potřebného DMA kanálu. Následuje systémové volání pro spuštění kanálu. V rámci

něj se kontroluje, zda již kanál neběží. Pro jistotu se tu podruhé ujišťuje, že je tento kanál využíván exkluzivně. Po tomto kroku již následuje samotné přijímání nebo odesílání dat.

Při odesílání dat se v předešlém návrhu aplikace snažila uzamknout nějaký počet paketů, který chtěla poslat. Důvod toho byl opět systém odběratelů. Tím, že s kanálem mohlo pracovat více procesů, nebyla jistota, že aplikaci bude vyhověno. V nové implementaci toto z důvodu exkluzivního využívání kanálu zaručeno je a proto je zamykání vynecháno. Zároveň se v tomto bodě vykonávala synchronizace s ovladačem, což bylo úplně odstraněno a knihovna obstarává ukazatele sama. Adresy míst pro pakety se tu před přenosem stále kopírují do pole struktur aplikace, protože pro změnu tohoto principu by musela být změněna práce aplikace, což je nežádoucí. Zároveň zde nelze použít platforma DPDK, protože tento modul DMA nepodporuje paketové přenosy. Do bufferu k DMA přenosu se před paket vždy vkládají hlavičky, které jsou pro řadič nezbytné. Následuje oznámení hardwaru, že jsou dostupné nové pakety. Původně se s ním čekalo, až bude zaplněna alespoň čtvrtina bufferu ovladače. V nové verzi je o tom řadič informován po každém burstu, zapsáním softwarového ukazatele do komponenty otevřené při spouštění kanálu. Pro příjem dat je postup velmi podobný. Původní zamykání se zde také nepoužívá, ale je nahrazeno kontrolováním, zda se nenavýšil hardwarový ukazatel, což by znamenalo, že přišly nové pakety. Narozdíl od Tx směru se tu ale data již nekopírují a místo toho jsou aplikaci akorát předávány adresy, kde se přijaté pakety nacházejí.

Poslední částí při přesunu dat pomocí tohoto modulu je zastavení DMA kanálu. Toto chování je zachováno pomocí systémového volání, ale s rozdílem toho, že jsou kernelu předány aktuální hodnoty ukazatelů z user space. Je důležité na konci přenosu oznámit ovladači stav, ve kterém se kanál nachází, aby při inicializaci další aplikace měl aktuální informace. Se zastavením kanálu je třeba i uzavřít komponentu otevřenou pro zápis softwarového ukazatele a tím práce končí.

5.2 Medusa

U paketového přenosu DMA je potřebné analyzovat práci s hardwarem podrobněji. Činnost začíná při vkládání modulu do systému. K tomu aby nový návrh zajišťoval komunikaci s kernelem bez použití ovladače, je třeba dostat některé interní data z kernel space do user space. Konkrétně tyto data jsou hardwarové ukazatele (hdp a hhp), pole deskriptorů a pole fyzických adres. Proto je potřeba, stejně jako u SZE, využít systémového volání `mmap`. To znamená, že při inicializaci modulu jsou pro potřebné pole alokovány místa v paměti a pak jsou tyto místa zaregistrovány pro `mmap`. Každé z nich je přiřazen název a offset ve stromu zařízení. To je z toho důvodu, aby když pak bude knihovna chtít mapovat tyto data do své paměti, mohla zjistit jaký offset má použít. Pro pole, které mohou mít proměnlivé velikosti je nutné do stromu přidat i jeho velikost, aby při mapování byly namapovány všechny položky.

S připraveným modulem už lze začít příprava přenosu. Opět je zachován postup z původní implementace a začíná se otevřením znakového zařízení a následným otevřením kanálu DMA. Nejprve probíhá inicializace v kernelu. Pomocí systémového volání je ovladačem otevřen kanál. Medusa zachází se systémem odběratelů stejným způsobem jako modul SZE, tedy otevírá kanál exkluzivně. V tuto chvíli si stejně jako v modulu SZE knihovna namapuje potřebné položky do své datové struktury. Jsou to:

- Paketový buffer.
- Hardwarové ukazatele.

- Pole fyzických adres.
- Pole deskriptorů.

Pole fyzických adres, stejně tak jako paketový buffer, ale nemusí být získány od ovladače. Tím, že knihovna nově podporuje platformu DPDK, mohou tyto dvě pole být v aplikaci (konkrétně DPDK aplikaci) již dostupné. Pole fyzických adres slouží k vytváření deskriptorů v průběhu přenosu dat. Otevření kanálu tedy ve výsledku primárně vytváří datovou strukturu reprezentující DMA kanál, kterou navrací aplikaci.

Startování DMA kanálu začíná otevřením komponenty kanálu pro pozdější zápisy softwarových ukazatelů, stejně jako v SZE. Pro start se používá systémové volání `ioctl`. V rámci něj ovladač nejprve kontroluje zda kanál neběží kvůli exkluzivitě. Pak se provádí inicializace hardwaru. Ovladač registry naplní počátečními hodnotami a podle typu běhu se chování ovladače větví. Typy běhu v tomto případě představují, zda se jedná o Rx nebo Tx kanál a zda jsou paketový buffer a fyzické adresy předány knihovně od kernelu nebo jsou již dostupné od aplikace (DPDK). Jestliže knihovna nevyužívá DPDK, je v tomto bodě pole fyzických adres vytvořeno takto:

1. Postupně se projde celý paketový buffer.
2. Z každého místa pro paket se získá jeho fyzická adresa.
3. Fyzická adresa je uložena do pole fyzických adres.

Pokud je pole fyzických adres již poskytnuto od aplikace, je tento krok vynechán. To ukončuje práci ovladače a následuje návrat do user space. V něm tímto práce Tx končí a kanál je připravený k přenosu. Pro Rx zbývá poslední krok. Když potřebuje knihovna odeslat data, vytváří si deskriptory pro každý paket průběžně. Když chce ale data přijmout, musí být deskriptory připravené dříve, než vůbec aplikace o pakety zažádá. Proto musí být před přenosem pro Rx vytvořeny deskriptory, které pak bude DMA moct využívat. Tento postup je poměrně jednoduchý:

1. Zkontroluje se, zda již nebyly deskriptory vytvářeny a podle toho nastaví se index v poli fyzických adres.
2. Získá se adresa indexovaná v poli fyzických adres.
3. Tato fyzická adresa je porovnána s fyzickou adresou posledního vytvářeného deskriptoru. Konkrétně se porovnává pouze vrchních 34 bitů těchto adres (jestliže se jedná o první vytvářený deskriptor, tak porovnání selže).
4. Vytvoří se jeden ze dvou deskriptorů:
   - Pokud porovnání projde úspěšně, vytvoří se deskriptor typu 2 a velikost v něm je nastavena na maximální možnou (v tomto bodě nelze znát velikost přijímaného paketu). Zároveň je inkrementován index v poli fyzických adres.
   - Pokud porovnání selže, vytvoří se deskriptor typu 0 a index se nezvyšuje.

Jestliže je takto vytvářeno více deskriptorů, je tento postup cyklem od bodu 2 do bodu 4. Tato funkcionalita je převzata z ovladače zařízení s rozdílem získávání paměti. Při inicializaci Rx je tedy téměř naplněno pole deskriptorů. Konec bufferu je ponechán volný pro

nově vytvářené deskriptory. Tím uvedení kanálu DMA do provozu končí. V tuto chvíli již lze pro oba směry provozu přenášet data.

Běh rozdělím opět na přijímací a odesílací část, protože mají v určitých ohledech rozdílnosti. Konkretně v tomto bodě jsou tyto rozdíly nové chování, probíhající místo synchronizace s kernelem a použití ukazatelů. Je dobré opět začít odesílací stranou kvůli jednoduchosti. Modul Medusa sice využívá jiné funkce pro posílání dat, než modul SZE, přesto je velká část tohoto procesu velmi podobná. Nejdříve byla snaha uzamknout potřebný počet paketů pro přenos. Tento krok je odebrán kvůli jeho zbytečnosti v novém návrhu. Na tomto místě se větví chování podle druhu aplikace. Pokud je použito DPDK, je fyzická adresa každého paketu nahrána do pole fyzických adres v knihovně pro pozdější použití. Jestliže se jedná o standardní aplikaci (bez použití DPDK), je tento krok vynechán, protože jsou fyzické adresy jsou již dostupné. Následně musí být pro standardní aplikaci adresy míst pro pakety předány aplikaci, což pro DPDK neplatí. Na tyto adresy pak může aplikace vložit své data. Tímto jsou data připravena k přenosu. Pro odeslání dat musí aplikace použít další funkci, která slouží k uvolnění zamčených paketů. Ta původně prováděla primárně synchronizaci s kernelem, kde se děl zbytek práce. V novém modelu je toto prováděno v knihovně. Předtím než je hardwaru oznámeno, že jsou dostupné nové pakety k odeslání, musí pro ně být vytvořeny deskriptory. Ty jsou vytvářeny velmi podobným způsobem, jako je při inicializaci Rx směru:

1. Zkontroluje se zda již nebyly deskriptory vytvářeny a podle toho nastaví se index v poli fyzických adres.

2. Získá se adresa indexovaná v poli fyzických adres.

3. Tato fyzická adresa je porovnána s fyzickou adresou posledního vytvářeného deskriptoru. Konkrétně se porovnává pouze vrchních 34 bitů těchto adres (jestliže se jedná o první vytvářený deskriptor, tak porovnání selže).

4. Pokud porovnání selže, vytvoří se deskriptor typu 0 a index se nezvyšuje. Jestliže porovnání prošlo úspěšně, tento krok se přeskakuje.

5. Vytvoří se deskriptor typu 2 a v něm je velikost nastavena na velikost paketu podle jeho hlavičky. Zároveň je inkrementován index v poli fyzických adres.

Po vyrobení potřebného počtu deskriptorů se za každý vyrobený inkrementuje sdp. Ukazatel sdp se nakonec zapisuje do otevřené komponenty. Tak se DMA dozví, že může přenášet data. Tímto přenos pro Tx končí, a jestliže chce aplikace posílat další pakety, stačí je znovu pouze připravit a odeslat.

Příjem je v mnoha ohledech opět velmi podobný. Stejně tak jako v Tx byla první část zamčení potřebného počtu paketů. Tento úsek je upraven tak, aby sloužil pro pouze nastavení potřebných proměnných a zjištění, zda nejsou dostupné nové pakety. Pokud nejsou, funkce končí. Pokud jsou, získají se jejich potřebná metadata z hlaviček. Pro standardní aplikaci je nutné aby knihovna předala aplikaci adresy, kde se pakety nacházejí. Data už jsou sice aplikaci dostupná, ale je potřeba ještě provést práci s deskriptory, kterou měl původně na starosti ovladač. Je stejně tak jako v Tx využito navracení paketů. V tomto případě se pakety neodesílají, ale pouze se připravují deskriptory pro nově příchozí data. V rámci toho je potřeba zjistit kolik deskriptorů bylo použito a kolik jich je třeba recyklovat. S tím pomáhá hardware, který do hlaviček paketů přidává informaci o tom, zda pro daný paket byl využit jeden, nebo dva deskriptory. Průchodem hlaviček všech paketů se dá postupným

sčítáním zjistit, kolik je třeba recyklovat deskriptorů. Nicméně aby tím nebyl přenos zatěžován neustále pro každý paket, jsou deskriptory vytvářeny pouze pokud je nevyužit počet o velikosti burstu (standardně 64) a více. Deskriptory jsou vyráběny stejně tak, jak bylo popsáno v inicializační části. Jestliže nějaké byly vyrobeny, jsou sdp a shp inkrementovány a zapsány do hardware, aby řadič věděl, že má nové deskriptory k dispozici. Zapisuje se opět do komponenty otevřené na začátku. Pro přijetí dalších paketů se dá tento proces opakovat stejným způsobem jako u Tx.

Už zbývá jen ukončení přenosu. To začíná zastavením kanálu DMA. Postup je tu téměř identický jako u SZE. Nejdříve je uzavřena komponenta kanálu v knihovně, následně je nastaven softwarový ukazatel pro kernel a nakonec je voláno systémové volání pro zastavení kanálu. Ovladač pak akorát musí zkontrolovat, zda jsou ukazatele konzistentní s hardwarem a nakonec zastavit kanál zápisem do řídícího registru. Mezi poslední akce knihovny patří uvolnění dat a uzavření zařízení. Tím celá práce končí.

Kapitola 6

Testování a měření

Pro jednoduchou práci s kartami je vytvořena aplikace ndptool. Její součástí jsou různé nástroje pro kontrolní odesílání a přijímání dat. Aplikace dokáže vytvářet pakety k přenosu i zpracovávat příchozí. Je schopna i obojího zároveň a pokud jsou data pomocí lokální smyčky přesměrovávána z aplikace opět na příjem, dá se tato vlastnost použít i pro testování celé cesty paketu. Jako API používá knihovnu libnfb a pomocí ní ovládá komponenty karty. Dokáže generovat různé množství paketů o rozdílných velikostech. Samotná data obsahují informace a vytvořeném paketu, jako je například identifikátor kanálu, ze kterého odchází, pořadí paketu v rámci burstu, velikost paketu a další. Tyto hodnoty se pomocí hashovací funkce transformují na hodnotu, kterou je pak naplněn paket. Pro větší pakety se hodnota opakuje po celou délku dat. Pomocí tohoto mechanismu lze kontrolovat, zda přenos dat probíhá v pořádku a že nejsou pakety po cestě modifikovány, nebo ztráceny.

Pro testování změn v knihovně a ovladači zařízení byla vybrána právě tato aplikace. Pomocí lokální smyčky byly generované pakety pouštěny systémem přes Tx do Rx. Na příchozí straně bylo zjišťováno, zda je vše v pořádku a zda změna obsluhy nepřinesla nezamýšlené následky. Pro měření v aplikaci ale neexistovala žádná funkcionalita a proto musela být aplikace upravena, aby měření poskytovala. Takto modifikovaná pak byla v rámci měření také využita.

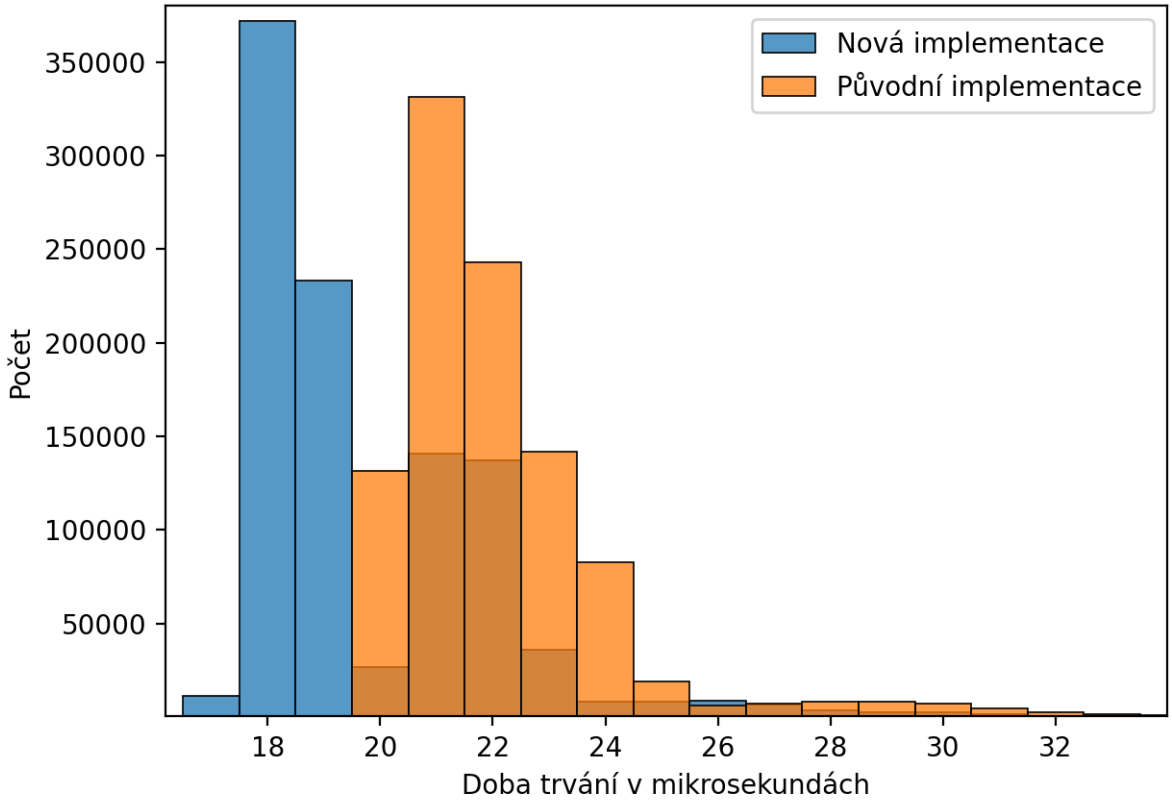
6.1 Úprava aplikace pro měření

Aby se dalo zjistit trvání přenosu paketů, je nutné zjistit aktuální čas při výrobě paketu a později získat čas při čtení. Když budu tyto dva časové údaje mít, získám dobu trvání odečtením prvního časového údaje od druhého (v pořadí, v jakém jsou získány). Pro jeden paket by tento proces byl velice jednoduchý a šlo by jej udělat pouze v rámci aplikace bez dalších úprav. Když je ale zapotřebí zjistit dobu v systému pro několik paketů, situace se komplikuje. Nejjednodušším způsobem by bylo možné uchovávat původní časy v poli. Tam ale nastává problém, že přenášených paketů může být obrovské množství, pro což by jednoduché pole nemuselo být dostačující. Kvůli tomu byl navržen systém, ve kterém se časová značka, kdy byl paket vyroben, vloží do jeho samotných dat. V tomto případě je tato značka vložena na bezprostřední začátek dat a není v průběhu paketu opakována. Když paket dorazí na výsledné místo, stačí pouze zjistit aktuální čas a vyčíst časovou značku z paketu, z nichž je vypočtena doba přenosu. To znamená, že není třeba uchovávat jakoukoliv informaci v aplikaci. Zároveň to znamená, že se dá testovat i doba přenosu mezi

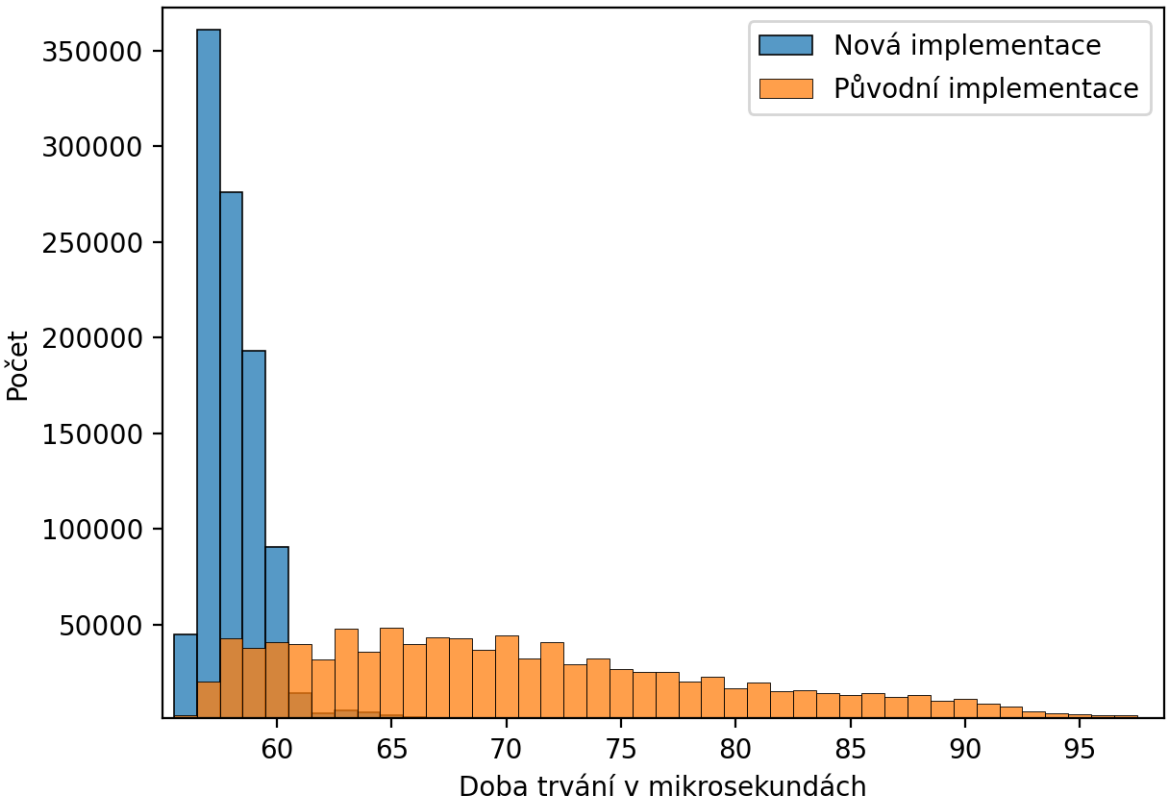
různými fyzickými stroji, pokud budou tuto aplikaci používat obě strany. Takto je měření prováděno pro jeden paket.

6.2 Výsledky měření

Měření bylo prováděno pomocí zmíněné aplikace. Nebyl zde zvolen postup jako v sekci 4.2, aby výsledky demonstrovaly efekt změn na celý přenos. Konkrétně jsem zvolil způsob, kdy je nástroj spouštěn vždy pro přenos jednoho paketu o velikosti 64 bytů. Byla povolena lokální smyčka a odesílaná data se ihned posílala zpět na příjem. Na příjmu byla také spuštěna aplikace a ta získávala dobu přenosu paketu. Aby byly výsledky statisticky významné, zvolil počet běhů aplikace na jeden milión. Tedy je získán milión časových údajů, které jsou zaneseny do grafu a zobrazeny v podobě histogramu. Byly takto testovány oba dva moduly s tím, že každý je umístěn do svého vlastního grafu.



Obrázek 6.1: Výsledky pro DMA SZE



Obrázek 6.2: Výsledky pro DMA Medusa

	SZE		**Medusa**	
	Původní	**Nová**	**Původní**	**Nová**
Aritmetický průměr	22,15	19,8	70,8	58,1
Geometrický průměr	22,1	19,69	70,2	58,08
Medián	22	19	69	58

Tabulka 6.1: Významné hodnoty obou měření zaokrouhlené na dvě desetinná místa

6.3 Hodnocení výsledků

Začnu výsledky modulu SZE zobrazenými v grafu 6.1. Je vidět, že se v novém návrhu dosáhlo nižší latence, než v původním. Konkrétně v nejlepším případě až o 3 mikrosekundy. Průběh celým systémem se v novém návrhu pohybuje kolem 18 mikrosekund. Nicméně kolem původních 22 mikrosekund je stále lokální maximum. Z hodnot tabulky se tedy dá říct, že celkové zlepšení je přibližně o 3 mikrosekundy.

Pro modul Medusa v grafu 6.2 je výsledek o něco lepší. Latence nové implementace jsou výrazněji více soustředěné na nižší hodnoty a jsou tedy obecně menší. Latence dosahuje nejčastěji 58 mikrosekund, což je oproti původní implementaci až o 11 mikrosekund rychlejší. Nejrychlejší přenos zůstává pořád stejný a tedy 56 mikrosekund (z grafu není úplně patrné), ale v novém návrhu této latence dosahuje výrazně více hodnot. V tomto případě o zlepšení napovídají jak oba průměry, tak i medián.

Kapitola 7

Závěr

Úmyslem práce bylo navrhnout způsob, kterým bude ušetřen čas při přenosu síťových dat, mezi uživatelskou aplikací a akcelerační kartou. Podmínkou bylo zachovat strukturu aplikací využívající knihovnu zastřešující tento proces. Pro dosažení tohoto cíle bylo nutné nejdříve nastudovat základní principy Linuxových operačních systémů, přiblížit si problematiku hardwarové akcelerace v rámci DMA přenosů a hlavně se seznámit s platformami vyvíjenými sdružením CESNET. Před návrhem bylo nutné podívat se na již existující způsoby těchto přenosů, přiblížit si jejich obecné funkcionality a kriticky je shrnout.

Samotný návrh se pak skládal z vyhrazení hlavního cíle a některých omezení, nalezení problémových částí softwaru a následnou eliminaci zpomalujících prvků. V implementaci pak byly řešeny různé úpravy návrhu, aby byl výsledný software kompatibilní s oběma druhy DMA modulů. Ukázalo se, že navržený způsob bylo možné pro oba moduly i zcela realizovat. Na závěr bylo důležité zjistit, zda takto implementované ovládání přenosu je ve výsledku rychlejší, než byl původní systém, což bylo měřením potvrzeno.

Literatura

[1] Bovet, D. P. a Cesati, M. *Understanding the Linux Kernel*. 3. vyd. O'Reilly Media, Inc, 2006. ISBN 978-0-596-00565-8.

[2] Corbet, J., Rubini, A. a Kroah Hartman, G. *Linux Device Drivers*. 3. vyd. O'Reilly Media, Inc, 2005. ISBN 0-596-00590-3.

[3] Neha, T. *Direct Memory Access (DMA)* [online]. 2019 [cit. 2022-06-05]. Dostupné z: https://binaryterms.com/direct-memory-access-dma.html.

[4] Documentation. *DPDK Project* [online]. 2022 [cit. 2022-09-04]. Dostupné z: http://core.dpdk.org/doc/.

[5] Yemelianov, A. *Introduction to DPDK: Architecture and Principles* [online]. 2016 [cit. 2022-9-04]. Dostupné z: https://blog.selectel.com/introduction-dpdk-architecture-principles/.

[6] Høiland Jørgensen, T., Brouer, J. D., Borkmann, D., Fastabend, J., Herbert, T. et al. The EXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel. In: *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. New York, NY, USA: Association for Computing Machinery, 2018. CoNEXT '18. DOI: 10.1145/3281411.3281443. ISBN 9781450360807. Dostupné z: https://doi.org/10.1145/3281411.3281443.

[7] Kotásek, Z. *Studijní opora předmětu Periferní zařízení.* [online]. FIT VUT v Brně, 2021 [cit. 2022-13-03]. Dostupné z: https://www.fit.vutbr.cz/study/courses/IPZ/public/.

[8] Red Hat. *What is the Linux kernel?* [online]. 2019 [cit. 2022-26-03]. Dostupné z: https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel.

[9] Red Hat. *What is Linux?* [online]. 2019 [cit. 2022-26-03]. Dostupné z: https://www.redhat.com/en/topics/linux/what-is-linux.

[10] *What You Need to Know to Get Started With Linux* [online]. 2020 [cit. 2022-26-03]. Dostupné z: https://comptechdoc.org/.

[11] Liberouter. *Liberouter Gitlab* [online]. 2022 [cit. 2022-22-04]. Dostupné z: https://gitlab.liberouter.org/ndk/swbase.

[12] UKEssays. *History of Network Interface Cards* [online]. 2018 [cit. 2022-8-04]. Dostupné z: https://www.ukessays.com/essays/information-technology/the-history-of-the-network-interface-cards-information-technology-essay.php.

[13] Scholz, D. *A Look at Intel's Dataplane Development Kit* [online]. 2014 [cit. 2022-09-04]. Dostupné z: https://www.net.in.tum.de/fileadmin/TUM/NET/NET-2014-08-1/NET-2014-08-1_15.pdf.

[14] Priya, M. *Virtual Memory* [online]. 2022 [cit. 2022-27-03]. Dostupné z: https://teachcomputerscience.com/virtual-memory/.

[15] Documentation. *Liberouter wiki* [online]. 2022 [cit. 2022-20-04]. Dostupné z: http://redmine.liberouter.org/projects/tmc/wiki.