VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE VIRTUÁLNÍHO PŘEPÍNAČE OPEN VSWITCH

ACCELERATION OF OPEN VSWITCH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DAVID VODÁK**
AUTHOR

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

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Vodák David**
Program: Informační technologie
Název: **Akcelerace virtuálního přepínače Open vSwitch**
Acceleration of Open vSwitch
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se s knihovnou DPDK, klasifikačním rozhraním RTE Flow, jazykem P4, virtuálním přepínačem Open vSwitch (OvS) a jeho možnostmi pro přesun klasifikačních pravidel na úroveň chytré síťové karty.
2. Seznamte se s platformou NDK, P4 kompilátorem a DPDK PMD ovladačem vyvíjeným v rámci sdružení CESNET.
3. Navrhněte vhodný způsob akcelerace OvS přepínače (DPDK) s využitím platformy NDK, technologie SR-IOV a jazyka P4.
4. Proveďte implementaci navrženého řešení a jeho funkčnost ověřte na dostupném hardware.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti dalšího pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

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 2019
Datum odevzdání: 28. května 2020
Datum schválení: 25. října 2019

Abstrakt

Virtuální přepínač je program, který slouží k přípojení virtuálních strojů k síti, a proto je velmi důležitou součástí virtualizace serveru. Nicméně virtuální přepínač spotřebovává značné množství výkonu serveru, na kterém běží. Pro virtuální přepínač Open vSwitch (OvS) bylo naměřeno, že při síťovém provozu o rychlosti 10 Gb/s spotřebuje přibližně 4 jádra procesoru. Spotřeba výkonu pak roste s rychlostí přenosu a eventuálně se může dostat do bodu, kdy bude neúnosná. Tato bakalářská práce, se zabývá akcelerací OvS za pomoci rozšíření DPDK Poll Mode Driveru, který OvS bude používat. Je zaměřena na rozšíření DPDK PMD o podporu virtualizační technologie SR-IOV a rozhraní pro offload klasifikačních pravidel do hardware RTE flow. V rámci práce byla implementována podpora SR-IOV v PMD a následně otestována na OvS. Dále byla navržena a částečně implementována podpora RTE flow.

Abstract

Virtual switch is a program, which is used for connecting virtual machines to network and that is why it is a crucial part of server virtualization. However virtual switch is consuming too much performance of the server which it is running on. A measurement of Open vSwitch (OvS) indicates that for data speed of 10 Gb/s, approximately 4 cores of the processor are fully occupied. As the consumption of performance is directly proportional to transmission speed, it may eventually get to the point where the consumption of performance cannot be handled. This bachelor thesis is about acceleration of the Open vSwitch with the help of the DPDK Poll Mode Driver extended by support of the SR-IOV virtualization technology as well as the interface for offloading classification rules to hardware called RTE flow. In the scope of this thesis the SR-IOV is implemented and then tested on OvS. Furthermore, the RTE flow support was designed and partially implemented.

Klíčová slova

DPDK, FPGA, hardwarová akcelerace, NDK Platforma, Open vSwitch, P4, RTE flow, síťová karta, SR-IOV

Keywords

DPDK, FPGA, hardware acceleration, NDK Platform, network interface controller, Open vSwitch, P4, RTE flow, SR-IOV

Citace

VODÁK, David. *Akcelerace virtuálního přepínače Open vSwitch*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Martínek, Ph.D.

Akcelerace virtuálního přepínače Open vSwitch

Prohlášení

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

.....
David Vodák
28. května 2020

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Tomáši Martínkovi, Ph.D. za věnovaný čas a odbornou pomoc. Dále bych rád poděkovat členům sdružení CESNET v aktivitě programovatelný hardware za odbornou pomoc a příjemné pracovní prostředí.

Obsah

Kapitola 1

Úvod

V dnešní době se čím dál více uplatňuje virtualizace, jelikož je levnější, rychlejší a dynamičtější než klasické používání serveru. Toto ale znamená, že na serveru existuje nějaký počet virtuálních strojů a je velice pravděpodobné, že zde neexistuje stejný počet síťových karet. Tento problém se dá vyřešit pomocí virtuálního přepínače. To je software, který dokáže přepínat pakety mezi virtuálními stroji a kartou. Tímto dokáže připojit všechny virtuální stroje k síti, i když je na serveru k dispozici jenom jedna síťová karta. Nicméně režie za používání virtuálního přepínače může být poněkud vysoká.

Proto se v této práci budu zabývat akcelerací virtuálního přepínače, kde se pomocí ovladače síťové karty dá přenést část zátěže z CPU do FPGA (Field-Programmable Gate Array) čipu, který je dostupný na kartě. Toto je realizováno jednak podporou virtualizační technologie SR-IOV a poté i offloadem klasifikačních pravidel do hardware.

Cílem této práce je seznámení s technologiemi nutnými pro již zmíněnou akceleraci. To konkrétně zahrnuje DPDK, klasifikační rozhraní RTE flow, jazyk P4, virtuální přepínač OvS, platformu NDK, P4 kompilátor, DPDK Poll Mode Driver vyvíjený v rámci sdružení CESNET a technologii SR-IOV. Následně je potřeba navrhnout vhodný způsob této akcelerace a implementovat jej.

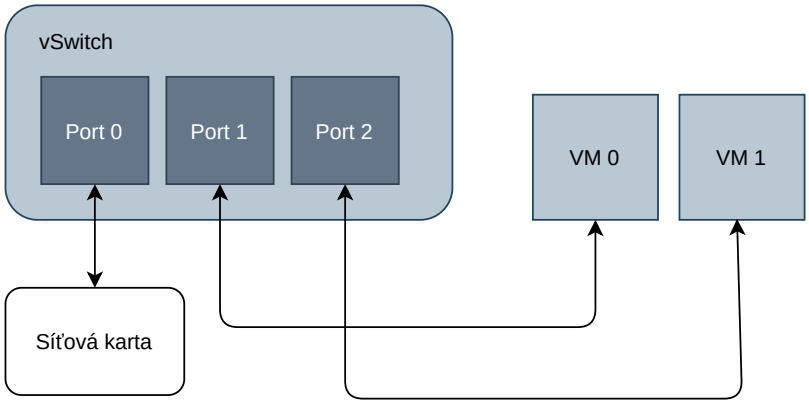
V rámci této práce se mi podařilo úspěšně navrhnout a implementovat rozšíření DPDK Poll Mode Driveru o virtualizační technologii SR-IOV. Dále jsem navrhl podporu RTE flow, a částečně ji implementoval.

Práce obsahuje 4 kapitoly. V následující kapitole jsou popsány klíčové technologie, které budou použity k akceleraci. Tato kapitola je rozdělena do následujících sekcí. V sekci 2.1 je popsán Open vSwitch, což je, jak už název napovídá, virtuální přepínač. Sekce 2.2 shrnuje a porovnává již implementovaná řešení akcelerace OvS. Virtualizační technologie SR-IOV je popsaná v sekci 2.3. Sekce 2.4 popisuje soubor knihoven DPDK. V sekci 2.5 je pak popsané RTE flow, což je DPDK rozhraní pro offload (přesun) klasifikačních pravidel do hardware. Jazyk P4, na kterém je má implementace založena, je popsán v sekci 2.6. Sekce 2.7 pak pojednává
o NDK platformě, do které mimo jiné patří i překladač jazyka P4 do VHDL a `libp4dev`, což je implementace vlastního P4 Runtime. V další kapitole je popsaná praktická část práce. Sekce 3.1 se zabývá návrhem pro rozšíření Poll Mode Driveru tak, aby pomocí rozhraní P4 Runtime byl schopen podporovat RTE flow a SR-IOV. Sekce 3.2 a 3.3 pak popisují testovaní a implementaci daného rozšíření. Poslední kapitola obsahuje závěr.

Kapitola 2

Teoretická část

V této kapitole jsou popsány technologie, potřebné pro akceleraci virtuálního přepínače. Před tímto popisem je však vhodné vysvětlit pojem virtuální přepínač. Virtuální přepínač je program, který přepíná pakety mezi virtuálními stroji (*virtual machine* – VM) a fyzickými porty, viz obrázek 2.1. Tím zajišťuje připojení virtuálních strojů k síti. To z něj dělá nedílnou součást virtualizačních technologií, bez které by existence virtuálních strojů na serveru nedávala smysl. Nicméně přepínání paketů mezi virtuálními stroji a fyzickými porty spotřebuje značnou část výkonu procesoru. To v praxi může znamenat několik jader, které jsou plně zaměstnány jen funkcí virtuálního přepínače. Z tohoto důvodu existuje spousta metod jak přenést zátěž z procesoru do síťové karty. Více ohledně této problematiky je popsáno v sekci 2.2.



Obrázek 2.1: Jednoduchý příklad virtuálního přepínače

2.1 Open vSwitch

V této sekci je popsán Open vSwitch a informace v této kapitole byly čerpány z oficiální dokumentace [6].

Open vSwitch, zkráceně OvS, je virtuální přepínač a jak už název napovídá, jedná se o open source software. Open vSwitch podporuje spoustu virtualizačních technologií jako například Xen/XenServer, KVM a VirtualBox. OvS může přistupovat ke kernel a DPDK zařízením, tyto zařízení je možné připojit na Open vSwitch porty. OvS pak mezi těmito

porty může přepínat pakety. Pro práci s OvS existuje spoustu příkazů, které jsou uvedeny níže.

Prvním z nich je `ovs-vswitchd`. Jedná se o démona, který řídí jednu nebo více instancí OvS, nicméně pouze jedna instance OvS může na stroji běžet v daný moment. Užitečné informace o běhu virtuálních přepínačů pak zapisuje do logu, který je uživateli k dispozici. Program `ovs-vswitchd` se při startu připojuje k databázi spravovanou `ovsdb-server`, tam se ukládají informace o konfiguraci OvS, které přetrvají i po konci běhu `ovs-vswitchd`.

Dalším nástrojem je `ovs-vsclt`, tento program slouží ke konfiguraci již zmíněné OvS databáze. Typickou konfigurací databáze může být například přidání portu do virtuálního přepínače. Portem se v tomto případě rozumí síťová karta, nebo její část (viz sekce 2.3). Kromě zapisovaní do databáze z ní `ovs-vsclt` může i vyčítat, aby se uživatel mohl ujistit, zda jsou porty připojeny ke správným zařízením.

Open vSwitch démoni přijímají určité příkazy za běhu. Tyto příkazy je možné poslat přes program `ovs-appctl`. Zároveň se dá `ovs-appctl` použít k získávání informací o OvS démonech. Typický příkaz je vypisování logů daného démona, kde si uživatel může nastavit, jaké typy logů chce vidět (od těch určených pro debug až po ty, které oznamují vážnou chybu).

Jako poslední bych rád zmínil `ovs-ofctl`. Tento příkaz slouží pro ovládání a administraci Open Flow přepínačů. Pomocí tohoto programu se dají vkládat klasifikační pravidla do určitých portů. Zároveň se díky tomuto programu dá zjistit počet už takto zadaných pravidel, nebo počet paketů, který od startu programu protekl virtuálním přepínačem.

2.2 Aktuální stav řešené problematiky

Jak už bylo zmíněno dříve v kapitole 2.1, přepínání paketů pomocí virtuálního přepínače má svou cenu. Konkrétně pro OvS bylo naměřeno, že při síťovém provozu o rychlosti 10 Gb/s spotřebuje Open vSwitch přibližně 4 jádra procesoru [12]. Tento problém je pak o to větší, vezmeme-li v úvahu větší rychlosti jako například 50 Gb/s, v takovém případě se stává režie OvS neúnosnou.

Tvůrci OvS si ale tuto skutečnost uvědomují a snaží se tento problém řešit. Prvním způsobem, jak zmenšit zátěž procesoru, je použití pokročilých virtualizačních technologií jako například SR-IOV. Pomocí této technologie je možné posílat pakety přes síťovou kartu přímo do virtuálního stroje, a tak obejít hypervizor. SR-IOV používám v rámci této práce a je popsané v sekci 2.3.

Další možností, jak ulehčit procesoru práci, je přenést pravidla z tabulek OvS do hardware a tam je implementovat. Zde existují dva způsoby, jak toto zařídit. Buď může jít o takzvaný částečný offload, kde se v hardware pakety pouze klasifikují, pak se označí a podle příslušné značky se v software provede požadovaná akce. Druhým způsobem je tzv. plný offload. V tomto případě se v hardware provádí jak klasifikace, tak příslušné akce. Tento způsob je efektivnější než předchozí, nicméně klade na hardware větší požadavky. U částečného offloadu totiž stačí vytvoření klasifikace a jedné akce na označení paketů pro celkovou podporu, zatímco u plného offloadu je potřeba podporovat širokou škálu akcí.

Jak plný, tak částečný offload může být proveden pomocí dvou různých rozhraní. V případě použití Linux kernel verze OvS je možné použít TC Flower. Pro OvS–DPDK pak existuje rozhraní RTE flow pro zajištění offloadu. Open vSwitch s využitím DPDK je rychlejší než tradiční verze postavená nad kernel rozhraním, nicméně podpora TC floweru

je u OvS stabilnější a lépe zdokumentovaná. Podpora plného offloadu pro RTE flow byla totiž přidána do OvS teprve na začátku roku 2020.

Dalším způsobem, jak zajistit offload, je implementace celého OvS v kartě. To je možné za předpokladu, že se v kartě nachází procesorová jádra, na kterých by Open vSwitch mohl běžet. V tomto případě žádné rozhraní pro offload pravidel není potřeba, jelikož celé OvS už v kartě běží.

Podporu těchto technologií už několik firem pro svoje síťové karty implementovalo. Firma Broadcom nabízí karty typu NetXtreme [10], které mají podporu pro SR-IOV a také plný offload OvS. V tomto případě běží celé OvS přímo v síťové kartě, jelikož v kartě existuje několik jader ARM, na kterých Open vSwitch může běžet. Při měření výkonosti u OvS bylo naměřena propustnost blížící se k 50 Gb/s [9].

Společnost Marvell nabízí chytré síťové karty typu LiquidIO, které podporují plný offload a SR-IOV [15]. Open vSwitch běží uvnitř těchto chytrých karet, podobně jako u společnosti Broadcom. Tyto karty obsahují procesorová jádra typu MIPS, díky kterým je toto možné. Výrobci při měření s OvS zjistili že propustnost se může pohybovat až okolo 35 Gb/s. Nicméně pro velmi malé pakety byla naměřena propustnost mezi 10 a 15 Gb/s [14].

Firma Mellanox nabízí síťové karty pod označením ASAP^2, což znamená Accelerated Switching and Packet Processing [16]. To už naznačuje, že mají jistou podporu pro virtuální přepínače. Konkrétně produkt ASAP^2 direct zajišťuje plný offload se zapojením SR-IOV a ASAP^2 flex zajišťuje offload částečný. Při měření bylo zjištěno, že u ASAP^2 direct je možné docílit propustnosti přibližně desetkrát větší než u DPDK verze OvS (což je nejvýkonnější OvS verze bez offloadu). U ASAP^2 flex byla naměřena propustnost v průměru 1,5 až 2,5 krát větší než u OvS s DPDK [17]. Tento offload je možný používat jak přes rozhraní TC flower, tak přes RTE flow.

Společnost Netronome vytváří karty typu Agilio, které jsou vybaveny síťovým procesorem s označením NFP [19]. Tyto síťové karty podporují full offload pro Open vSwitch. V měřeních provedené výrobcem jde vidět, že dosahují propustnosti až 40 Gb/s. Z grafů měření lze také vidět, že této rychlosti zpracování dosahují tyto procesory zpravidla u větších paketů. Jako rozhraní pro offload paketů používá Netronome TC Flower [20].

Firma Napatech nabízí karty s podporou akcelerace OvS, pro kterou nabízí dvě řešení. Prvním z nich je částečný offload, o kterém výrobci tvrdí, že je o 80 procent výkonnější než klasické OvS s DPDK bez offloadu. Další způsob je pak založen na používání vlastního rozhraní Direct VirtIO, kvůli kterému je nutné upravit samotné OvS. Nicméně díky tomuto je možné dosáhnout propustnosti 40 Gb/s na obou portech karty SmartNIC i na těch nejkratších paketech. U této karty je také důležité zmínit, že jako jediná z výše uvedených používá technologii FPGA[18].

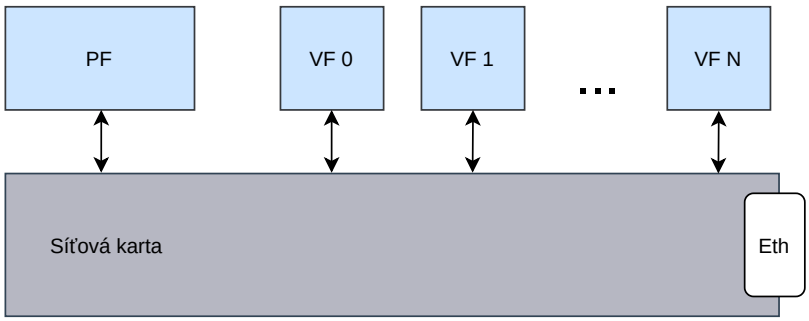
Řešení této problematiky v mojí práci je vyvíjeno v rámci sdružení CESNET a firmy Netcope Technologies, a. s. Je zde použito karet typu COMBO s čipem FPGA, což může být klíčové u provozu skládajícího se z malých paketů. Výše zmíněné měření ukazují, že u takových provozů se propustnost karet používající procesory zmenšuje. Proto se zde jeví jako výhodnější použití technologie FPGA. Jediná společnost, která se snaží o akceleraci OvS, s tímto typem karet je Napatech.

Ve srovnání s řešením firmy Napatech jsou karty typu COMBO ve výhodě z několika důvodů. Prvním důvodem je podpora technologii SR-IOV. Ta je, na rozdíl od Direct VirtIO, všeobecně známou technologií, která je podporovaná přímo v OvS a nemusí se kvůli ní Open vSwitch upravovat. Dalším důvodem je možnost plného offloadu, kterou společnost Napatech nenabízí. Jako poslední je důležité zmínit, že v mojí práci je vyvíjeno rozhraní

pro OvS s DPDK, což se projevuje jako rychlejší varianta než ta používající Linux kernel rozhraní.

2.3 SR-IOV

Single Root Input/Outuput Virtualization, zkráceně SR-IOV, je rozšíření specifikace PCI express (PCIe), které dokáže předat přístup více hardwarovým funkcím a rozdělit mezi ně zdroje fyzického zařízení. SR-IOV využívá koncept fyzické a virtuálních funkcí. Fyzická funkce, anglicky *Physical Function* (PF), má stejné vlastnosti v software jako jakékoliv jiné PCIe zařízení. Fyzické funkce tak mohou být použity na ovládání a konfiguraci daného PCIe zařízení (například síťové karty). Pro PCIe zařízení podporující SR-IOV existuje zpravidla jedna fyzická funkce. Virtuální funkce, zkráceně VF, oproti fyzické funkci nemůže být použita na konfiguraci protože nemá přístup do fyzického zařízení. Virtuálních funkcí bývá zpravidla více a jsou použity právě proto, aby bylo možné reprezentovat jedno fyzické zařízení více zařízeními v software. Tato virtualizace je většinou realizována v hardware, a proto musí být podporovaná jak ve zařízení tak v BIOSu. Dalším omezením této technologie je neschopnost měnit počet virtuálních funkcí za běhu aplikace [13]. Na obrázku 2.2 je schéma síťové karty s obecně N virtuálními funkcemi.



Obrázek 2.2: Síťová karta s podporou SR-IOV

Typickou aplikací, která tento koncept používá, je virtuální přepínač, který běží nad síťovou kartou s podporou SR-IOV. Na hardwarové funkce jsou pak napojeny buď virtuální stroje, nebo virtuální přepínač. Díky tomuto mechanismu je možné přenášet pakety přímo přes hardware a ušetřit tak výkon. Jinak by totiž virtuální stroje musely být připojeny přes hypervizor, což znamená zmenšení rychlosti a zvýšení požadavků na CPU. Popis implementace podpory SR-IOV pro DPDK ovladač je v sekci 3.2.

2.4 DPDK

Tato sekce pojednává o systému DPDK a většina informací je čerpána z oficiální dokumentace [3].

Data Plane Development Kit je, zkráceně DPDK, sada knihoven psaná v jazyce C, která umožňuje rychlé zpracování paketů. DPDK podporuje širokou řadu procesorů, mezi které patří procesory řady x86, POWER a ARM. Co se operačních systémů týče, DPDK je vyvíjeno primárně pro Linux, ale je spustitelné i ve FreeBSD, nově je také možné spustit jednoduchou ukázkovou aplikaci v systému Microsoft Windows. DPDK podporuje síťové karty od výrobců jako například Intel, Cisco, Mellanox, NXP, Netronome, Amazon a spousty

dalších. Mezi podporované síťové karty patří také karty typu COMBO, které jsou vyvíjeny společností Netcope a sdružením CESNET. Více informací je možné najít v kapitole 2.7.

DPDK se využívá pro vývoj síťových aplikací a je výhodné v situacích, ve kterých je požadována velká datová propustnost. Existuje několik důvodů proč tomu tak je, jedním z nich je například *polling*. *Polling* je princip, při kterém se aplikace používající DPDK neustále dotazuje, jestli ze síťové karty nepřišel paket. To znamená, že i při minimálním počtu příchozích paketů bude muset být pro tuto obslužnou rutinu jedno logické jádro. Výhodou tohoto principu je schopnost obejít se bez přerušení, které by musely být vyvolány po každém příchozím paketu. Přerušení trvá řádově několik desítek taktů, což je dost velká režie, vezmeme-li v potaz maximální počet příchozích paketů u dnešních vysokorychlostních síťových karet.

Dalším důvodem, proč je DPDK výhodné pro velký datový provoz, je možnost použití velkých stránek (*hugepages*), jak už název napovídá, jedná se o velké tabulky stránek. Normální tabulky mají velikost 4 KB (v případě architektury x86 a operačního systému Linux), což při nutnosti ukládat větší množství paketů za jednotku času často způsobuje TLB miss a to zpomaluje cílovou aplikaci. *Hugepages* tento problém řeší, jelikož to jsou velké stránky o velikostech daleko větší než 4 KB (v operačním systému Linux), například 2 MB nebo jeden 1 GB. Při takto velkých stránkách je zde mnohem méně překladů mezi virtuální a fyzickou adresou, a tím pádem to zmenšuje i šanci na TLB miss. Více informací o *hugepages* lze nalézt v [11].

DPDK se skládá z mnoha částí, přičemž několik hlavních knihoven, které tvoří základ DPDK, se nazývají Core Components. Jednou z nejdůležitějších částí DPDK je Enviroment Abstraction Layer, zkráceně EAL. Je to vrstva, která zapouzdřuje nízkoúrovňové zdroje a poskytuje rozhraní pro práci s nimi, jedná se například o zprávu logických jader nebo ovladačů. Mbuf library je knihovma, která poskytuje datovou strukturu `mbuf`. Struktura `mbuf` slouží pro uložení paketu, přičemž jeden `mbuf` může pojmout maximálně jeden paket. Další důležitou částí je Ring library, ta poskytuje datovou strukturu `rte_ring`, což je kruhový seznam `mbuf` struktur. Tato struktura pomáhá přenášet `mbufy` z ovladače do aplikace.

Pro podporu karet v DPDK je nutné vytvořit DPDK ovladač, který bude poskytovat rozhraní pro knihovny DPDK a bude umístěn v složce drivers v hlavním DPDK adresáři. Takový ovladač se na nazývá Poll Mode Driver, což se zkracuje na PMD. Poll Mode Drivery běží v user space, jinými slovy nemůžou využívat kernel rozhraní jako normální ovladače. Tím pádem využívají kernel moduly a knihovny, které jsou schopny poskytnout rozhraní pro práci s kartou. PMD tedy obaluje rozhraní již existujícího ovladače ve funkcích tak, aby mohl fungovat s DPDK. Ukazatele na tyto funkce jsou při inicializaci zapsány do struktury `rte_dev_ops`, díky čemuž tyto funkce pak můžou používat DPDK knihovny. Rozhraní DPDK knihoven pak už může využívat konkrétní DPDK aplikace.

V DPDK je síťová karta reprezentována strukturou `rte_eth_dev`. Aplikace k této struktuře přistupuje nepřímo pomocí indexu do funkcí knihoven DPDK. Daná funkce už pak podle indexu najde tu správnou `rte_eth_dev` strukturu a vykoná nad ní patřičnou akci. Typicky se z dané struktury získá ukazatel na funkci, aby DPDK mohlo používat funkce daného ovladače. Předtím, než aplikace bude takto používat rozhraní ovladače, je nutné inicializovat kartu a daný ovladač. Toto se děje automaticky při startu aplikace, kdy DPDK zavolá ze struktury `rte_pci_driver` callback `probe`, který se postará o veškerou inicializaci. Jde primárně o registraci operací, které ovladač potřebuje k tomu, aby k nim mohla přistoupit aplikace přes DPDK API, jak je uvedeno výše. Dále jde o inicializaci zdrojů karty, jako jsou například DMA Kanály. Interní informace o ovladači, které jsou

například spojeny s touto inicializací, se pak ukládají do struktury, na kterou lze dosáhnout přes `rte_eth_dev`. Tento odkaz je realizován pomocí ukazatele na *void*, což umožňuje každému ovladači definovat svou vlastní strukturu.

Jedněmi z nejdůležitějších funkcí, které jsou dostupné z `rte_eth_dev`, jsou funkce pro přenos paketů. Jedná se buď o funkci pro příjem (*rx*), nebo vysílání (*tx*) paketů. Pokud chce uživatel zavolat tyto funkce ovladače, použije na to rozhraní DPDK. V tomto rozhraní se specifikuje jak index portu (`rte_eth_dev` struktury), tak index fronty. Fronty se v DPDK používají právě pro posílání paketů a každé zařízení potřebuje alespoň jednu frontu, aby mohlo být použito pro síťový provoz. Nicméně front může mít zařízení i více. Před použitím se ale fronta musí inicializovat pomocí DPDK rozhraní, které následně volá funkci ovladače, kde se inicializace provádí.

DPDK je také schopno podporovat technologii SR-IOV, která je popsaná v sekci 2.3. Činí tak tím, že vytvoří více struktur `rte_eth_dev` pro jednu síťovou kartu. Jedna z nich bude hlavní struktura, zastupující kartu samotnou. Ostatní budou reprezentovat SR-IOV virtuální funkce a podle toho jsou pojmenovány jako reprezentátory. Reprezentátory nemají vlastní PCI zařízení, a tím pádem ani strukturu `rte_pci_driver`, díky kterému by mohly být inicializovány. Z tohoto důvodu jsou vytvořeny v rámci inicializace hlavní `rte_eth_dev` struktury. Kromě faktu, že reprezentátory nemají svoje vlastní fyzické zařízení, se nijak od jiných `rte_eth_dev` struktur neliší. Dají se rozpoznat podle názvu nebo podle flagu RTE_ETH_DEV_REPRESENTOR. Moje konkrétní implementace podpory SR-IOV v DPDK Poll Mode Driveru je popsaná v rámci sekce 3.2.

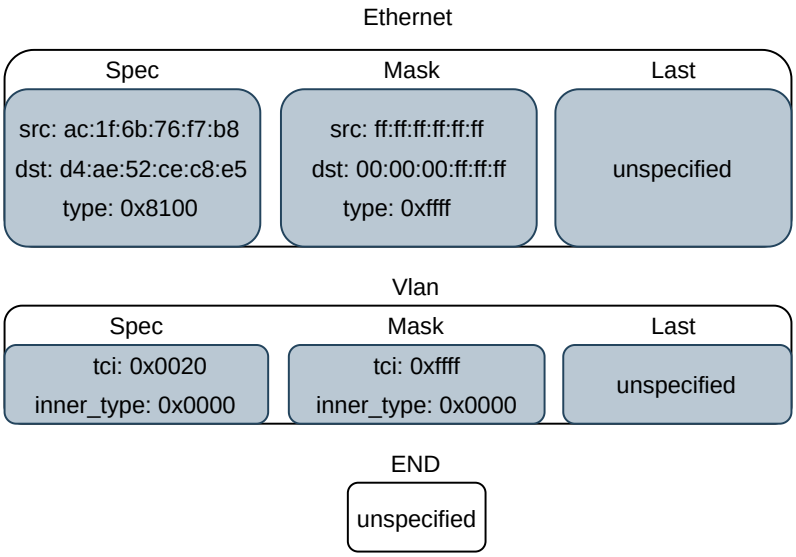
2.5 RTE flow

V této sekci je popsán RTE flow, což je DPDK rozhraní pro offload klasifikačních pravidel do Hardware. Informace uvedené v této kapitole jsou převzaty z oficiální DPDK dokumentace [5].

RTE flow k definici pravidla pro klasifikaci paketů (*flow rule*) používá atributy (*attributes*) daného pravidla, dále shodný vzor (*matching pattern*) a jako poslední akce (*actions*), které se mají nad paketem vykonat. Zjednodušeně řečeno, RTE flow zajistí, aby se nad pakety, které jsou specifikovány *matching pattern*, provedly *actions*.

Atributy pravidla, které jsou reprezentovány jako struktura `rte_flow_attr`, jsou doplňující informace o pravidlu. Skupina (*Group*) patří mezi tyto atributy a umožňuje volit číslo skupiny daného pravidla. Pokud má skupina číslo nula, nic se nemění a význam pravidla je stejný jako předtím. Ovšem pokud pravidlo bude mít číslo skupiny větší než 0, aplikuje se pouze na základě akce JUMP. Tato akce slouží právě k tomu, aby přesměrovala paket do akcí s daným číslem skupiny. Dalším atributem je priorita (*Priority*), která je reprezentována jako přirozené číslo. Čím menší toto číslo je, tím vyšší je priorita pravidla. V případě, že by paket vyhovoval více pravidlům, zvolí se to s nejvyšší prioritou a aplikuje se. Směr provozu (*Traffic direction*) je další atribut, ten definuje, kde se paket bude klasifikovat. Zde se může nastavit, aby se paket klasifikoval při vstupu do pipeline (*ingress*), při výstupu z pipeline (*egress*) nebo v obou případech (*both*). Přenos (*Transfer*) je poslední z atributů, daný atribut dovoluje přesunout provoz mimo aplikaci, pokud tento síťový provoz do aplikace nepatří (může patřit například jiné aplikaci nebo virtulální funkci).

Matching pattern dovoluje specifikovat paket a je tvořena polem, které tvoří struktury `rte_flow_item` (zkráceně pouze *item*). Každý z těchto *itemů* zpravidla reprezentuje protokolovou vrstvu, nicméně jsou zde i výjimky. Například VOID, což značí prázdný item, PORT_ID, který značí číslo DPDK portu (index `rte_eth_dev` struktury), na kterém byl



Obrázek 2.3: Ukázka Matching pattern

paket přijat, nebo END, který slouží jako zarážka. Item, který reprezentuje protokolovou vrstvu, obsahuje části *spec*, *mask* a *last*. Parametr *spec* definuje hodnotu, která se bude porovnávat. Část *mask* definuje bitovou masku, která určí, jaké bity ze spec budou porovnávány. Například u Ethernetové položky na obrázku 2.3 je zdrojová adresa definována jako ac:1f:6b:76:f7:b8. Nicméně u cílové MAC adresy jsou specifikovány poslední 3 byty jako ce:c8:e5 a zbytek této adresy může být libovolný. Pokud je zadaný parametr *last*, *matching* se rozšíří z rozsahu od *spec* do *last*.Toto se pokusím vysvětlit na následujícím příkladu.

Příklad 2.1: Mějme *item*, kde je specifikovaná pouze cílová IPv4 adresa. U této adresy je *spec* 10.0.0.1 a *mask* není specifikovaný. Pokud není zadaný ani *last*, *item* se shoduje pouze s pakety, kde je cílová IPv4 adresa 10.0.0.1. Jestliže *last* zadaný bude, například na hodnotu 10.0.0.255, bude *item* specifikovat všechny pakety se zdrojovou IPv4 adresou v rozsahu 10.0.0.1 až 10.0.0.255.

Je třeba ještě dodat že *item* specifikuje jak zdrojovou, tak cílovou adresu a většinou i pár dalších polí v protokolové hlavičce. Na tomto příkladě zde byla pro jednoduchost uvedena pouze jedna adresa (toho lze prakticky dosáhnout tím, že se ostatní pole v hlavičce vymaskují). V obrázku 2.3 je pro lepší představu ukázána vícepoložková *matching pattern*.

Akce prováděná nad paketem je reprezentovaná strukturou `rte_flow_action`, ta má dvě položky. První z nich je typ (*type*), která určuje typ dané akce (například může mít hodnotu SET_MAC_DST). Druhá položka slouží pro konfiguraci (*conf*), v ní jsou uloženy potřebné informace pro vykonání dané akce (u předchozího příkladu by to byla specifikace cílové MAC adresy). Takových akcí může být specifikováno více, a proto jsou akce předávány jako pole těchto struktur zakončené položkou END, která slouží jako zarážka. RTE flow podporuje akce pro modifikaci hlaviček jako například již zmíněná SET_MAC_DST, kromě toho definuje akce pro přesměrování provozu typu PORT_ID. Tato akce přesměruje pakety specifikované *matching pattern* do jiného DPDK portu. Jako poslední akci bych chtěl zmínit akci COUNT. To je akce, pomocí které je možné nastavit čítač pro dané pravidlo. Z tohoto čítače je pak možné vyčíst, kolikrát se pravidlo nad pakety aplikovalo (*match*).

K zahájení offloadu se používá funkce `rte_flow_create()`, která jako argumenty bere již zmíněné pravidla, akce a `matching pattern`. V RTE flow API se pak nachází

také funkce `rte_flow_query()`. Ta slouží pro vyčítání informací, které poskytly pravidla, to je důležité zejména u akce COUNT. Zde se pomocí této funkce zjišťuje hodnota, kterou čítač daného pravidla napočítal.

2.6 Jazyk P4

Programming Protocol-independent Packet Processors, zkráceně P4, je jazyk o jehož vývoj se stará P4 Language Consortium. Informace v této sekci budou čerpány z jejich oficiální specifikace [7].

P4 je deklarativní jazyk, který je určen pro popis způsobu, kterým se budou zpracovávat pakety. Takový popis se pak může uplatnit například v přepínačích, směrovačích nebo síťových kartách. Tento popis pomocí jazyka P4 se dá uplatnit i u karet typu COMBO. Více o této technologii lze nalézt v sekci 2.7.

Jazyk P4 je založený na takzvaných *match-action* tabulkách. *Match* část určuje, jaké položky se u paketu budou porovnávat. *Action* část tabulky specifikuje, jaké akce by se nad paketem provedly, pokud by paket vyhovoval požadavkům *match*. Toto všechno musí být specifikováno před startem aplikace v rámci P4 zdrojových souborů. Co všechno se dá popsat pomocí tohoto jazyka je popsáno v následujícím odstavci.

Jako první je potřeba definovat hlavičky síťových protokolů, příklad takové definice se dá najít ve výpisu 2.1. Zde je vidět, že každý protokol se dá definovat podle potřeb programátora, dokonce se pomocí tohoto jazyku dá nadefinovat hlavička nového nebo neexistujícího protokolu. Po definici hlaviček je potřeba definovat parser. Jedná se o komponentu, která rozkládá paket na jednotlivé části a definuje se jako stavový automat. Typicky prvním stavem parsování je extrakce Ethernetové hlavičky, další stav je rozhodnut na základě informací získaných z dané hlavičky. Může to být například extrakce hlavičky IPv4, IPv6, VLAN nebo cokoliv za stav, který zde programátor napsal. Po parseru je nutné definovat *match-action* tabulky, které jsem již zmiňoval. Zde se musí definovat *match* část pomocí již definovaných hlaviček a poté akce. Akce se definují pomocí takzvaných primitivních akcí, kam patří například `modify_field()`, `add_to_field()` nebo `drop()`. Jako poslední je potřeba vytvořit takzvaný `control` blok, kde se určuje, jakým způsobem se budou dané tabulky aplikovat. V tomto bloku se dá větvit pomocí příkazu `if`, což dovoluje výběr tabulek na základě informací získaných z paketů v parseru. Zde může například být tabulka, která má na starosti výpočet IPv4 checksum, a ta bude aplikována, pouze pokud paket hlavičku IPv4 obsahuje.

Po definování struktury tabulek je potřeba do nich nahrát data. Toto je možné vykonat za pomoci rozhraní P4 Runtime. Přes toto rozhraní se pak dá vytvořit pravidlo (*rule*), a nahrát ho do tabulky, které zde doplní konkrétní data. Data týkající se části *match* se nazývají klíče (*keys*) a data pro doplnění informací o akci se nazývají parametry. Pravidla se pomocí P4 runtime přidávají až po startu aplikace a je jich možno přidat pro každou tabulku hned několik. Tuto funkcionalitu se pokusím vysvětlit na následujícím příkladu.

Příklad 2.2: Mějme tabulku, která má jako *match* položku definovanou zdrojovou IPv4 hlavičku typu *exact* a jako akci zde má změnu cílové MAC adresy. Typ *exact* znamená, že se bude porovnávat celá hodnota klíče, více o typech klíčů je popsáno níže. Po startu aplikace bylo vloženo pravidlo přes P4 Runtime, které bude mít jako klíč IPv4 adresu 147.229.3.10 a jako parametr bude mít MAC adresu 00:11:17:02:A1:23. Pro pakety, které budou mít jako zdrojovou IPv4 adresu 147.229.3.10, se změní cílová MAC adresa na 00:11:17:02:A1:23.

Existují různé typy P4 klíčů, typ klíče může být buď *exact*, *lpm* nebo *ternary*. Toto se specifikuje už v P4 zdrojových souborech. Jakým způsobem funguje typ *exact* už zde

```
header_type ethernet_t {
    fields {
        dst_addr :   48;
        src_addr :   48;
        ether_type : 16;
    }
}

header_type vlan_t {
    fields {
        pcp :        3;
        cfi :        1;
        vid :        12;
        ether_type : 16;
    }
}
```

Výpis 2.1: Ukázka definice hlaviček v jazyce P4

bylo demonstrováno. Tento typ je ze velmi jednoduchý, ale má nízké nároky na cílovou architekturu. Typ *lpm* ještě specifikuje prefix, který určuje, jak velká část z položky *match* se bude porovnávat. Pokud by výše v příkladu byl klíč typu *lpm* a hodnota prefixu byla 24, změnila by se cílová MAC adresa u všech paketů se zdrojovou IPv4 adresou, kde první tři oktety budou 147.229.3 a poslední oktet by mohl být libovolný. Jde vidět, že pomocí typu *lpm* se dá specifikovat mnohem více než pomocí *exact*, to však také znamená vyšší požadavky na cílovou architekturu. Posledním typem je *ternary*, zde se navíc specifikuje bitová maska, která určuje, jaké bity se budou porovnávat. Pokud by v již zmíněném příkladu byl klíč typu *ternary* a maska by měla hodnotu 255.0.0.255, byla by změněna cílová MAC adresa pro pakety se zdrojovou MAC adresou, kde první oktet je 147 a poslední oktet je 10. Typ *ternary* je výrazově mocnější než *lpm* nebo *exact*, nicméně o to náročnější na implementaci v hardware je.

2.7 NDK platforma

NDK (Netcope Development Kit) je platforma určená k síťové akceleraci za pomocí síťových karet obsahujících FPGA čip. Tyto technologie jsou vyvíjeny sdružením CESNET v rámci aktivity Programovatelný hardware a společností Netcope Technologies a.s. Ve své práci využívám karty typu COMBO, podpory jazyka P4 a DPDK PMD ovladače pro již zmíněné karty a spoustu dalších věcí z této platformy. Proto je popis těchto technologií nezbytný. Informace z této sekce jsou čerpány z manuálu firmy Netcope Technologies a.s.[8].

Karty typu COMBO jsou karty vyvíjené v rámci NDK platformy a obsahují moderní FPGA čipy od firmy Xilinx. Tyto karty dosahují propustnosti až 200 Gb za sekundu a dají se připojit na PCI Express třetí generace. Kartu COMBO-200G2QL je možné zapojit až do dvou takových PCIe portů, jedná se totiž o kartu s maximální propustností až 200 Gb/s, na což by jeden PCIe port třetí generace nestačil. Informace o těchto kartách jsou dostupné v [1]. Karta COMBO-200G2QL je také použita při testování mojí implementace praktické části této práce a je vidět na obrázku 2.4. Pro přenos paketů z karty do paměti se používá DMA (Direch Memory Access) kanálů. Těchto kanálů poskytuje karta zpravidla víc a každý z nich se dá používat odděleně. Kanály se dělí podle směru, kterým přesouvají

pakety. Buď může jít o kanály, které z karty přijímají pakety směrem do paměti (*rx*), nebo o kanály, které se starají o posílání paketů do karty (*tx*).

Obrázek 2.4: Karta COMBO-200q2ql s využitím jednoho PCIe portu

Pro tyto karty existuje Linux kernel ovladač *nfb* (Netcope FPGA Boards), který poskytuje rozhraní pro práci s kartou. Toto rozhraní pak může být využito pomocí knihovny `libnfb` přímo aplikací nebo linuxovým jádrem pro zapojení karty do TCP/IP stacku. V rámci NDK platformy existuje sada příkazů, která může být použita pro práci s kartou. Tyto příkazy obsahují buď prefix *nfb* nebo *ndp* (Netcope Data Plane). Nachází se zde příkaz pro nahrávání FPGA designů do karty (`nfb-boot`), což umožňuje změnu chování karty na základě nahraného designu. Chování karty se dá popsat pomocí jazyka VHDL a ze zdrojových souborů v tomto jazyce se dá vytvořit design pomocí syntézních nástrojů. Dále jsou zde příkazy pro vyčítání informací o kartě jako například `nfb-info`. Pomocí nich se dají zjistit informace ohledně nahraného designu, statistiky o přijatých a odeslaných paketech nebo teplota FPGA čipu. Jako poslední bych chtěl zmínit příkazy `ndp-transmit` a `ndp-read`. Jak už název napovídá, jedná se o příkazy pro příjem a vysílání paketů. Tyto příkazy jsou skvělou pomůckou pro testování, jelikož je možné poslat/přijmout pakety bez nutnosti vytváření vlastní aplikace. Zároveň je zaručeno, přesně jaké pakety se do karty posílají, což se například u komunikace přes TCP/IP stack těžko zjišťuje.

Další využití *nfb* ovladače a *libnfb* existuje v podobě DPDK Poll Mode Driveru, což zajišťuje podporu COMBO karet v DPDK. V DPDK existují dva Poll Mode Drivery podporující COMBO karty, kterými jsou *szedata* a *nfb*. PMD *nfb* je novější a používá stejnojmenný kernel ovladač. Na rozdíl od *szedata*, *nfb* umožňuje zapnutí hardwarové podpory SR-IOV. Z toho důvodu jsem si zvolil tento ovladač pro implementaci, která je popsaná v praktické části. Dál tedy popíšu pouze *nfb* Poll Mode Driver. Nicméně tyto ovladače jsou v principu dost podobné, až na to, že *nfb* používá novější kernel ovladač a nabízí pár nových vlastností.

Ve zdrojových kódech *nfb* Poll Mode Driveru se nachází instance struktur typu `rte_pci_driver` a `eth_dev_ops`. Zde jsou uloženy odkazy na všechny důležité funkce tohoto ovladače, které pak DPDK může zavolat. Struktura, do které se v tomto ovladači ukládají privátní data, se nazývá `pmd_internals` a je možné na ni přistoupit

z `rte_eth_dev`. Do `pmd_internals` se pak ukládá odkaz na strukturu `nfb_device`, která slouží jako hlavní *handle* (reference na objekt) pro práci s *libnfb*. Co se nastavení front týče, PMD *nfb* používá na každou frontu jeden DMA kanál. Tudíž zařízení disponuje tolika frontami, kolik DMA kanálů daná karta poskytuje.

Další část NDK platformy, která je velice podstatná pro tuto práci, je P4 překladač. Jedná se o program, který je schopen přeložit P4 zdrojové soubory do jazyka VHDL. Jelikož se dá pomocí syntézních nástrojů vytvořit design z VHDL zdrojových souborů, dá se tím pádem vytvořit design i na základě jazyka P4. P4 překladač tak poskytuje vysokoúrovňový popis hardware, díky čemuž je možné popsat chování karty i bez hlubokých znalostí návrhu hardware. Nicméně jsou zde i určité limity, dané fyzickými zdroji FPGA čipu. Takové limity lze například vidět u výběru typu porovnávacího klíče. Při použití nejnáročnějšího typu *ternary* je možné vložit pouze řádově desítky až stovky P4 pravidel. U typu *lpm* je situace o něco lepší, nicméně i zde jdou limit FPGA čipu vidět. Pro vyřešení těchto problémů je pak nutné implementovat podporu externích pamětí.

Pro komunikaci s P4 designem v aplikaci je možné využít NDK implementaci P4 Runtime jménem `libp4dev`. Pomocí této knihovny se dají vkládat pravidla do tabulek, stejně jako to bylo popsáno v sekci 2.6. Knihovna `libp4dev` poskytuje datový typ `p4device_t`, který se používá jako *handle* pro užívání rozhraní této knihovny a symbolizuje jednu P4 pipeline. Před jakýmkoliv používání karty s P4 designem je nutné inicializovat tento *handle* pomocí funkce `p4device_init()`, pak je vhodné P4 pipeline resetovat, aby se vymazala pravidla, která zde mohla být uložena od posledního běhu aplikace. Následně už je možné pracovat s kartou, nicméně je nutné zajistit, aby žádné funkce z této knihovny nebyly používány současně.

Jako poslední bych chtěl zmínit podporu SR-IOV, kterou kernel ovladač *nfb* nabízí. Ovládá se pomocí zápisu počtu virtuálních funkcí do souboru `sriov_numvfs`. Tento soubor se nachází někde v adresáři `/sys`, kde přesně už záleží na používaném linuxové distribuci, pro nalezení tohoto souboru je možné použít například příkaz *find*. Po zapsání do tohoto souboru se vytvoří nová zařízení, která jsou viditelná ve složce `/dev`, a dá se k nim přistupovat. Původní zařízení, které existovalo i před zapsáním do tohoto souboru (`/dev/nfb0`), se tak stává fyzickou funkcí a ostatní nově vzniklé zařízení jsou virtuální funkce. Toto vše je samozřejmě možné, pouze pokud je v kartě nahrán design, který SR-IOV podporuje. Je třeba ještě dodat, že počet virtuálních funkcí je zde limitován. Maximální počet těchto funkcí je sedm, to je dáno omezeným počtem DMA kanálů. Každé virtuální funkci je totiž přidělen jeden *rx* a jeden *tx* DMA kanál.

Kapitola 3

Praktická část

V této kapitole je popsán návrh, implementace a testování podpory RTE flow a SR-IOV v DPDK Poll Mode Driveru *nfb*. Po rozšíření tohoto PMD je možné jej využít v OvS, kde akceleruje jeho chod díky novým možnostem tohoto ovladače. Open vSwitch pak může využít podporu SR-IOV, která mu umožňuje připojit virtuální stroje přes hardware a ušetřit tak výkon. Následně pak může využít podporu RTE flow pro offload klasifikačních pravidel do hardware, což přenese část zátěže z CPU do hardware.

Pro zajištění těchto funkcionalit je nutné nejprve vytvořit firmwarovou podporu v FPGA čipu karty COMBO. Zde je nutné vytvořit mapování mezi jednotlivými hardwarovými funkcemi SR-IOV a následně implementovat podporu RTE flow. Poté je potřeba rozšířit *nfb* PMD tak, aby mohl využívat nově vytvořenou podporu ve firmware.

3.1 Návrh

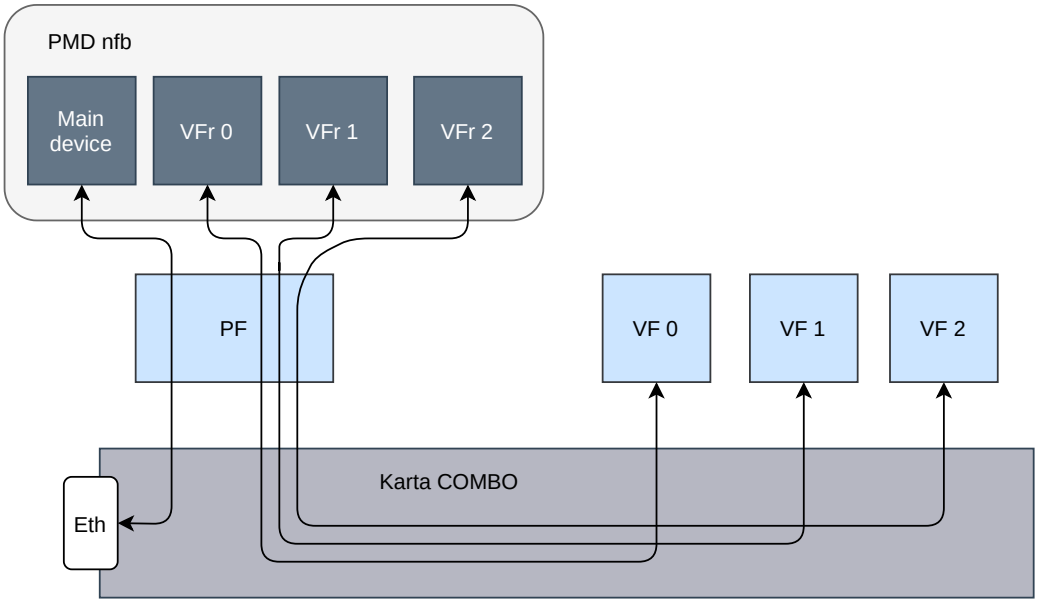
Jako první bych chtěl v rámci návrhu popsat cílovou architekturu, která tvoří OvS používající *nfb* PMD s SR-IOV a RTE flow. Využití SR-IOV hardwarových funkcí bude následovné. Fyzickou funkci bude využívat Poll Mode Driver a virtuální funkce budou využity virtuálními stroji. Uvnitř *nfb* PMD se pak vytvoří hlavní zařízení (*device*) a reprezentátory virtuálních funkcí. Virtuální funkce pak budou přes kartu napojeny na svoje reprezentátory, díky kterým bude moct OvS komunikovat s virtuálními stroji (přes tyto virtuální funkce). Hlavní zařízení bude připojeno k ethernetovému portu, což zajistí připojení OvS k síti.

Co se RTE flow podpory týče, je nutné zajistit podporu pro akce, které OvS je schopno offloadovat. OvS má ve zdrojových kódech podporu pro akce PORT_ID, COUNT, DROP, RSS a dále akcí pro změnu položek v protokolových hlavičkách paketu. Ze sledování činnosti OvS vyplynulo, že OvS se aktivně snaží o offload akcí typu PORT_ID, COUNT, DROP a RSS. V rámci implementace jsem se pak zaměřil všechny akce, které mají podporu v OvS, kromě RSS, jelikož pro tuto akci zatím neexistuje hardwarová podpora. Zde jsem ale kladl důraz na akce PORT_ID, COUNT a DROP, jelikož o jejich offload se OvS aktivně snaží. Je zde ale potřeba podotknout, že plný offload u OVS–DPDK je novou záležitostí, která je ještě ve vývoji a je možné, že počet podporovaných akcí poroste. Stejně tak lze předpokládat, že poroste počet akcí, které se bude snažit Open vSwich aktivně offladovat. Tato podpora RTE flow akcí bude realizována pomocí čipu FPGA v kartě, stejně jako zapojení SR-IOV hardwarových funkcí.

Pro vytvoření designu do FPGA čipu, byl využit jazyk P4. Pomocí P4 bude tedy řešeno přeposílání paketů mezi virtuálními funkcemi a jejich reprezentátory v případě podpory SR-

IOV. Zároveň je pomocí tohoto jazyka možné implementovat podporu RTE flow, jelikož z P4 primitivních akcí se dají vytvořit RTE flow akce. Z těchto důvodů v P4 designu existují dvě hlavní tabulky, které toto zajišťují. Zdrojové kódy, ze kterých je tento design vysyntetizován, byly vytvořeny v rámci sdružení CESNET a na jejich tvorbě jsem se částečně podílel.

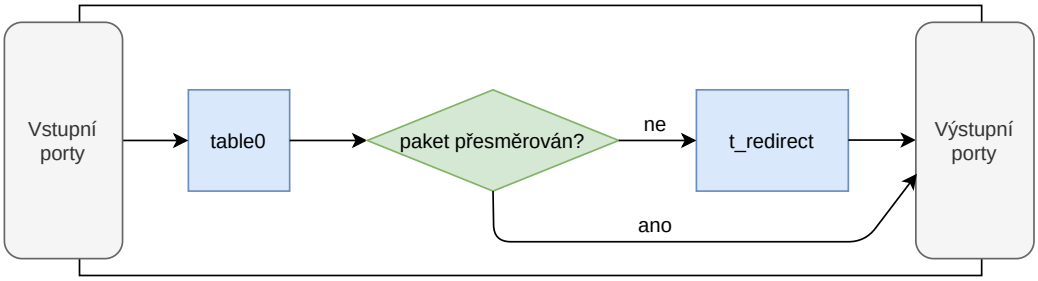
Tabulka `t_redirect` slouží pro přesměrování paketů do správných portů, aby bylo zajištěno správné zapojení SR-IOV hardwarových funkcí. Portem se rozumí buď DMA kanál a nebo Ethernetový port. V této tabulce se jako klíč používá číslo portu, ze kterého paket vyšel, a akce je změna portu, do kterého paket bude nasměrován. V ovladači se potom dají vytvořit pravidla pro tuto tabulku, které zajistí propojení virtuálních funkcí a jejich reprezentátorů. Kromě tohoto propojení je zde nutné také propojit hlavní zařízení v Poll Mode Driveru a Ethernetový port. Pro lepší představu je toto zapojení zobrazeno na obrázku 3.1, kde se počítá se třemi virtuálními funkcemi.



Obrázek 3.1: Ukázka zapojení SR-IOV

Tabulka `table0` se snaží zajistit podporu RTE flow pravidel. V této tabulce existuje jedna velká akce, která podporuje dané RTE flow akce. Činí tak tím, že pro každou RTE flow akci existují P4 primitivní akce, které ji implementují. V případě offloadu pravidla do této tabulky se pak nad paketem všechny tyto P4 primitivní akce aplikují. Problémem ale je, že Open vSwitch vytváří pravidla, ve kterých vyžaduje jen některé akce, proto je nutné nějakým způsobem zajistit, aby bylo možné vybrat jen některé P4 primitivní akce, které se nad paketem provedou. Jazyk P4, konkrétně jeho verze 14, která je zde používaná, nepodporuje větvení na úrovni akcí. Kvůli tomu nemůže provádět jen některé P4 primitivní akce. To znamená, že jediný způsob, jak zajistit tuto selekci, je použití masek pro každou P4 primitivní akci. Masky u primitivní akce `modify_field()` určují, jaké bity se v daném datovém poli budou měnit. Takže pokud by uživatel chtěl offloadovat pravidlo pouze pro změnu cílové MAC adresy, měla by maska u této primitivní akce hodnotu FF:FF:FF:FF:FF:FF a všechny ostatní masky by byly nulové. I když je tímto způsobem možné zaručit selekci, znamená to, že ve skoro každém P4 pravidle bude existovat velký počet takto vymaskovaných akcí, které nemají žádný účel. Co se P4 klíčů týče, je zde zvolena široká škála položek v protokolových hlavičkách a jako typ je zvolen *ternary*. Tento typ

je zvolen, aby bylo možné rozhodnout, podle jakých položek se bude klasifikace provádět. Je zde tedy aplikován obdobný princip jako u akcí. Tabulka tímto dovoluje porovnávat na základě spousty různých hlaviček a dovoluje i výběr, podle kterých hlaviček se porovnávat bude. Nicméně tohle všechno znamená, že se použije spousta P4 klíčů takového typu, který je na architekturu nejnáročnější. Toto dost omezuje počet offloadovaných pravidel.



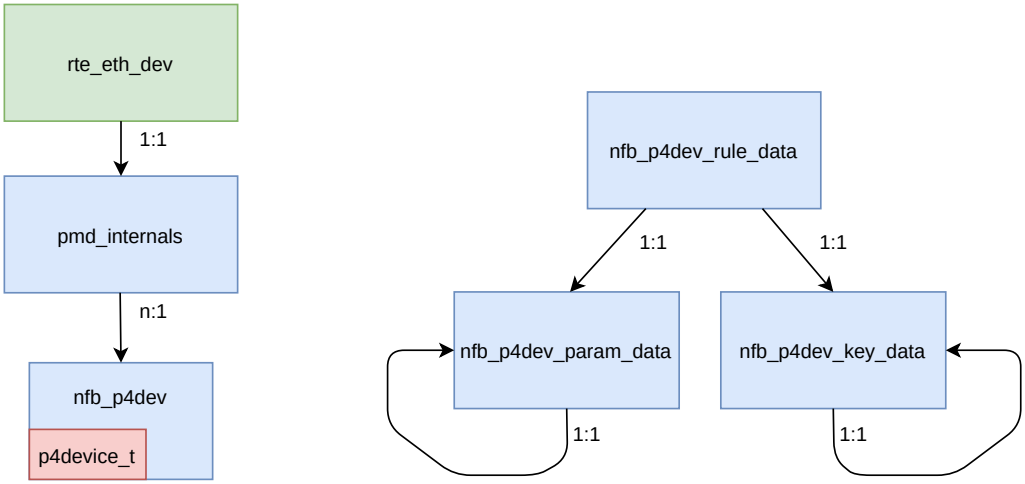
Obrázek 3.2: Zpracování paketů v P4 designu

Jedna z možných RTE flow akcí, které OvS požaduje, je akce PORT_ID, která přesměrovává pakety z jednoho portu do druhého. Z tohoto důvodu je nutné na pakety aplikovat prvně tabulku `table0`. Pokud se zde aplikuje akce PORT_ID paket už není dále přesměrováván a zamíří do nově nastaveného portu. Pokud ale tato akce aplikovaná není, paket zamíří do tabulky `t_redirect`, kde je nasměrován do cílového portu. Toto zpracování paketů v P4 designu je zobrazeno na obrázku 3.2

3.2 Implementace

Jak již bylo zmíněno v předešlé sekci, P4 design v kartě je nezbytný, a proto je nutné jej začít podporovat v DPDK Poll Mode Driveru *nfb*. K tomuto jsem využil knihovny `libp4dev`, kterou jsem do tohoto ovladače zaintegroval. Jelikož bude `libp4dev` volána jak kvůli offloadu RTE flow pravidel, tak kvůli SR-IOV, vytvořil jsem jednotné rozhraní nad touto knihovnou, které pak může být jednoduše voláno z obou částí ovladače. Toto rozhraní se nachází v odděleném souboru a kompletně zapouzdřuje práci s `libp4dev`. Jako *handle* se zde používá struktura `nfb_p4dev`, která obsahuje `p4device_t` a DPDK spinlock pro zajištění, že se žádná P4 funkce nebude provádět současně. Toto zamykání se děje na straně již zmíněného souboru, díky tomu se ve zbytku ovladače už tato problematika nemusí řešit. Na strukturu `nfb_p4dev` je pak možné přistoupit z odkazu v `pmd_internals`. Pro vytvoření pravidla je nutné naplnit strukturu `nfb_p4dev_rule_data`. Do této struktury je možné vložit proměnný počet P4 parametrů a P4 klíčů, jelikož jsou zde použity seznamy ze `sys/queue.h` knihovny. V obrázku 3.3 jsou vyobrazeny výše popisované struktury a vztahy mezi nimi. Po naplnění `nfb_p4dev_rule_data` je nad touto strukturou zavolaná funkce `nfb_p4dev_create_rule()`, která dané pravidlo offloaduje do karty. Samozřejmě je pak nutné paměť ze seznamů zase uvolnit, k tomu je možné použít funkci `nfb_p4dev_create_rule()`.

Jako první bych se chtěl věnovat podpoře SR-IOV, kterou jsem implementoval v *nfb* Poll Mode Driveru. Pro zajištění této podpory je nutné vytvořit v rámci jedné karty více zařízení, tak jak je to popsáno v posledním odstavci sekce 2.4. V případě PMD *nfb* jsou nová zařízení vytvářena v callbacku `probe`. Zde se podle *devargs* rozhodne, zda tato karta poběží v SR-IOV módu, nebo v módu klasickém. Rozhraní *devargs* umožňuje z příkazové řádky definovat počet reprezentátorů. Pokud je tento počet nenulový, ovladač poběží



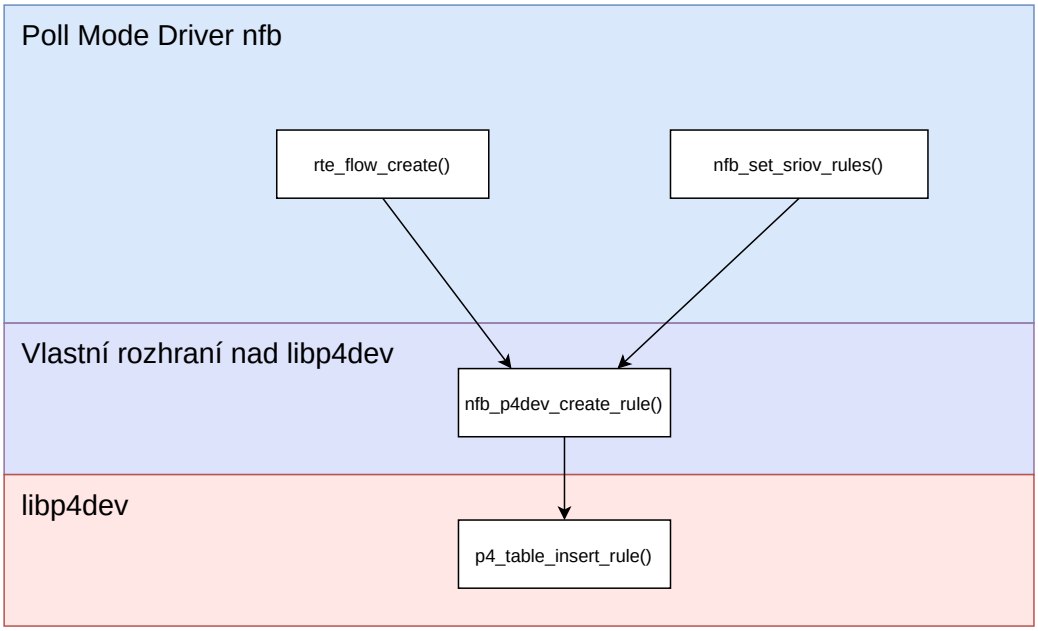
Obrázek 3.3: Vlastní struktury pro práci s `libp4dev`

v SR-IOV módu. V tomto případě se vytvoří daný počet reprezentátorů, pokud ovšem nepřesahuje maximální počet virtuálních funkcí podporovaný kartou. V takovém případě by ovladač zahlásil chybu, což by mělo za důsledek konec běhu aplikace. Mimo reprezentátory je v SR-IOV módu také vytvořeno hlavní zařízení, které zastupuje kartu samotnou.

I když jsou tato zařízení vytvořena a DPDK k nim může přistupovat, podpora SR-IOV pořád není kompletní. Je totiž nutné přidělit daným reprezentátorům DMA kanály, aby skrz ně mohly téct pakety. Každému reprezentátoru je nutné přidělit jeden *rx* a jeden *tx* DMA kanál, tak jako je to zobrazeno na obrázku 3.1. Zbytek DMA kanálů pak připadne hlavnímu zařízení. Inicializace DMA kanálů se v *nfb* Poll Mode Driveru provádí zavoláním funkce pro inicializaci fronty. Proto je nutné už před tím rozdělit DMA kanály mezi zařízení tak, aby všechny DMA kanály byly využity a žádný z nich nebyl používán více zařízeními. Toto rozdělení se děje v callbacku *probe*, kde se každému zařízení zapíše do `pmd_internals`, na jaké pozici leží jejich DMA kanály. Tento údaj se pak využije při nastavení fronty tak, aby byl inicializován ten správný DMA kanál.

Poté, co mají reprezentátory svoje DMA kanály, je potřeba tyto kanály namapovat na DMA kanály virtuálních funkcí tak, aby tvořily dvojice VF–VFr (reprezentátor virtuální funkce). Tyto dvojice je možné vidět na obrázku 3.1. Pro toto mapování jsem použil výše zmíněnou tabulku v P4 designu. Na straně Poll Mode Driveru je pak jen nutné zajistit nahrání patřičných pravidel pro přesměrování paketů do správných portů (portem se zde rozumí DMA kanál nebo Ethernetový port). Indexy portů dvojic jsou uloženy v poli struktur `redir_pairs`, ve kterém lze jednoduše dohledat, jaké porty jsou na sebe namapovány. Kromě dvojic VF–VFr je zde ještě uložen pár, který mapuje DMA kanály hlavního zařízení na Ethernetový port. V rámci funkce `nfb_set_sriov_rules()` je pak pro každou dvojici z pole `redir_pairs` zavolaná funkce `nfb_p4dev_create_rule()`, která pravidlo o tomto přesměrování offloaduje do hardwaru. Toto volání je vyobrazeno na obrázku 3.4.

Toto mapování ovšem funguje pouze pokud se v rámci inicializace vytvoří 7 reprezentátorů, což je maximální počet. Tyto reprezentátory ale musí být pouze vytvořeny, v rámci aplikace pak nemusí být aktivní, což znamená, že režie za tvorbu tolika reprezentátorů je skoro nulová. Jedinou nevýhodou je pak zbytečné obsazení DMA kanálů, nicméně tyto kanály by v PMD jinak neměly žádný účel.



Obrázek 3.4: Volání funkcí pro offload a jejich závislosti

Všechny tyto změny v *nfb* Poll Mode Driveru zajišťují podporu SR-IOV. Jako poslední bych pak chtěl zmínit, že všechny tyto zařízení mezi sebou sdílí odkaz na stejnou strukturu `nfb_p4dev`. Ta je totiž důležitá pro podporu RTE flow, která je popsána v dalším odstavci.

RTE flow je rozhraní, které umožňuje offloadovat až příliš mnoho typů akcí do hardware. Proto jsem do Poll Mode Driveru *nfb* přidal podporu pouze takových akcí, která jsou pro Open vSwitch nejdůležitější. A to jsou konkrétně akce PORT_ID, COUNT, DROP (akce RSS nebyla přidána z důvodu hardwarové nekompatibility) a následně akce, které mění hodnoty položek z L1, L2 a L3 hlaviček. Akce DROP způsobí zahození paketu a akce PORT_ID přesměruje provoz do jiného portu. V kombinaci s atributem *transfer* dokáže PORT_ID přesměrovat provoz z jednoho virtuálního stroje do druhého. Tato vlastnost je v rámci OvS velmi ceněná, jelikož se tímto odkloní veškerá zátěž daného provozu mimo virtuální přepínač. Akce COUNT se používá pro nastavení čítačů tak, aby bylo možné vyčítat z karty počty paketů, které vyhovují *matching pattern*. OvS tuto akci používá pro zjištění, zdá má smysl dané pravidlo offloadovat.

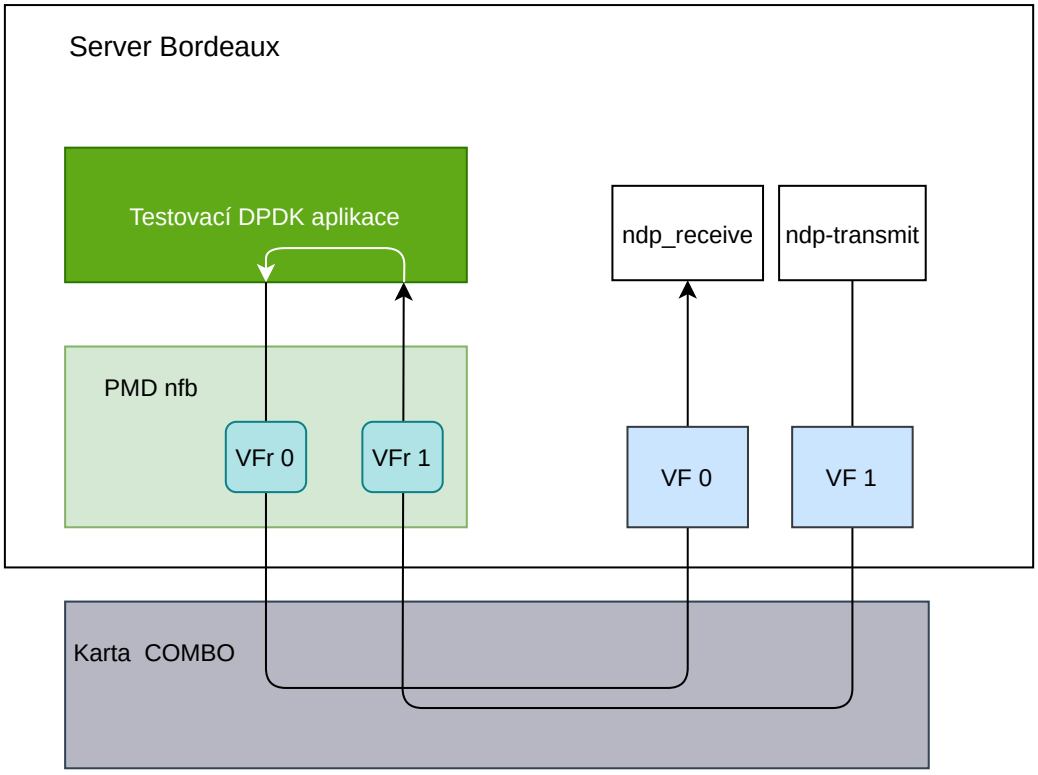
Pro implementaci podpory těchto akcí jsem použil již zmíněné rozhraní nad knihovnou `libp4dev`. Pro tvorbu pravidla používám šablonu (*template*), která obsahuje všechny parametry a klíče nutné pro offload pravidla. Tato šablona je ze začátku vynulovaná, což znamená, že jsou všechny políčka vymaskované a offload takové akce by neměl žádný vliv na síťový provoz. Ve funkci `nfb_flow_create()`, která slouží pro vytvoření RTE flow pravidla, se jako první vytvoří a vynuluje tato šablona. Následně se zjistí, jaké *itemy* obsahuje *matching pattern* a hodnoty (*spec* části) *itemu* se vloží do šablony. Pokud tyto *itemy* mají specifikovanou bitovou masku (*mask*), uloží se také do šablony, a pokud ne, do šablony se uloží maska, kde všechny bity budou mít hodnotu 1. Podobně se pak nastavují akce. Pro všechny RTE flow akce se uloží do šablony potřebné parametry a následně se nastaví bity masky na hodnotu 1. Po tomto zaplnění se ze šablony vytvoří P4 pravidlo, které se nahraje do karty.

Další RTE flow funkcionalitou, kterou tento ovladač poskytuje, je možnost využít funkce `nfb_flow_create()` pro odstranění pravidla a `nfb_flow_query()` pro zjištění hodnot čítačů v rámci akce COUNT. Všechny tyto funkce se pak dají zavolat přes rozhraní RTE flow. V rámci tohoto rozhraní se musí specifikovat dané zařízení pro offload. Jelikož je struktura `nfb_p4dev` sdílena mezi všemi zařízeními karty, dá se tak zavolat RTE flow funkce jak z reprezentátorů, tak z hlavního zařízení.

3.3 Testování

Tato sekce je rozdělena na dvě části, první z nich je zaměřena na testování podpory SR-IOV a ta druhá pojednává o testování RTE flow rozhraní Poll Mode Driveru. Testování probíhalo na serveru Bordeaux, který je vybaven kartou COMBO-200G2QL a operačním systémem Scinetific Linux.

Podporu SR-IOV jsem testoval různými způsoby. Prvním způsobem bylo použití vlastní jednoduché testovací DPDK aplikace. Tuto aplikaci jsem vytvořil podle příkladu *skeleton* [2]. Zde jsem měl možnost ověřit, že pakety cestují z virtuální funkce do reprezentátoru a opačným směrem. Aplikace funguje tak, že přijímá pakety z jednoho DPDK portu a posílá je do jiného portu. V kombinaci s příkazy `ndp-transmit` a `ndp-recieve` jsem mohl poslat paket z jedné virtuální funkce a přijmout ho pomocí druhé virtuální funkce. Tak jsem otestoval, že přes dvojice VF–VFr se dají posílat pakety a to hned oběma směry. Cesta paketu v tomto testu je naznačena v obrázku 3.5.



Obrázek 3.5: Cesta paketu v testu s DPDK aplikací

Jako další je pak ale nutné ověřit, zda podpora SR-IOV funguje i s OvS. Jelikož se ohledně OvS–DPDK nastavuje mnoho věcí, vytvořil jsem si složku se skripty, které tyto

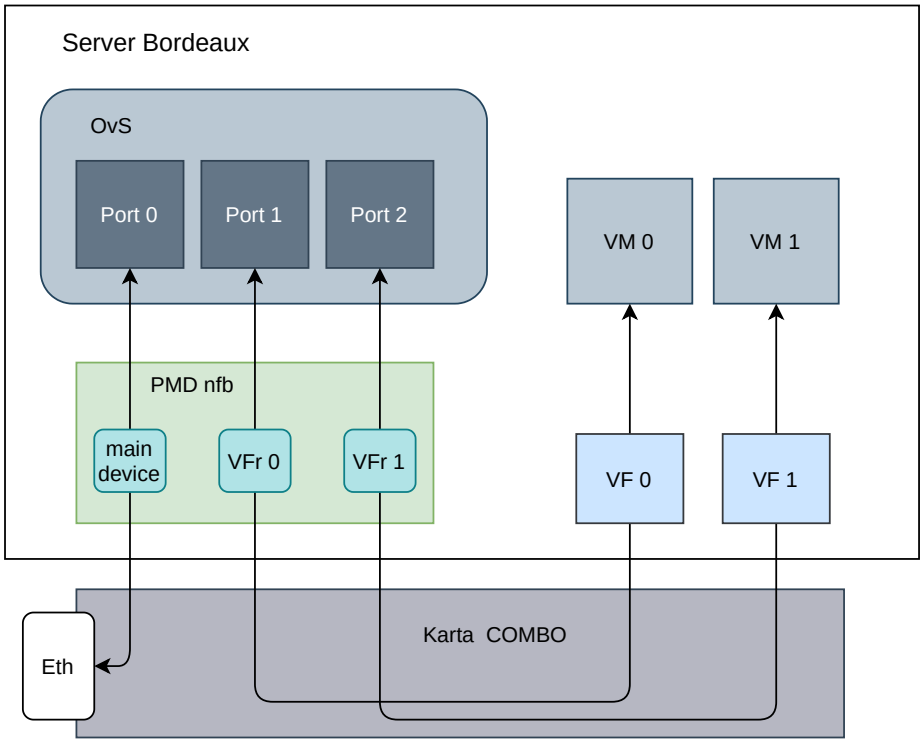
problémy řeší automatizovaně. Bylo potřeba nastavit server do stavu, kdy je možné spustit DPDK aplikaci (jako například OvS–DPDK) a to řeší skript `dpdk-setup.sh`. Tento skript zahrnuje nabootování karty a nastavení *hugepages*. Další důležité skripty, které se zde nachází, slouží k nastavení OvS tak, aby byl schopen pracovat s DPDK PMD karty COMBO a zařízeními, který tento Poll Mode Driver poskytuje. Zde nastává problém s identifikací těchto zařízení. V rámci OvS se totiž zařízení identifikují pomocí identifikátoru PCI portu. Nicméně reprezentátory v DPDK sdílí PCI port s hlavním zařízením a proto tato identifikace není dostačující. Proto OvS poskytuje možnost přidat k identifikátoru PCI portu ještě číslo, které identifikuje reprezentátor. Tato možnost, ale v mém případě nefungovala a proto jsem byl nucen toto obejít a identifikovat reprezentátory pomocí MAC adres po vzoru firmy Mellanox. Více o této pro problematice je možné najít v [4].

Tímto nicméně vznikl nový problém, jelikož v PMD *nfb* se MAC adresy náhodně generují (tedy alespoň jejich poslední 3 byty), a proto je nutné tyto MAC adresy při každém běhu OvS znovu zjišťovat. Tento problém jsem vyřešil úpravou Poll Mode Driveru *nfb*, ve které jsem zajistil vypisování MAC adres do logů. Tyto výpisy jsou pak promítnuty až do logu, který vypisuje `ovs-vswitchd`. Z tohoto logu jde tedy zjistit jaké MAC adresy k identifikaci použít. Pro zjednodušení tohoto procesu identifikace jsem napsal skript `add_representor.sh`, který zjistí MAC adresy z daného logu, a na základě nich vytvoří v OvS nové porty.

Postup při spouštění OvS byl následující. Jako první jsem spustil výše zmíněný skript `dpdk_setup.sh`. V tento moment je potřeba ještě vytvořit SR-IOV virtuální funkce pomocí skriptu `add_vfs.sh`. Následně jsem spustil `setup_ovs.sh`, který nastaví OvS pro práci s DPDK. Tento skript zároveň připojí první DPDK port, což je hlavní zařízení v PMD, které je možné připojit pouze pomocí identifikátoru PCI portu. V tento moment je OvS možné spustit pomocí příkazu `ovs-vswitchd`. Po zapnutí OvS se samozřejmě uvede do činnosti i Poll Mode Driver, a proto je možné zjistit MAC adresy reprezentátorů. K tomuto jsem použil již zmíněný skript `add_representor.sh`, který zároveň tyto reprezentátory použije k vytvoření OvS portů.

Nyní už je možné používat OvS–DPDK s SR-IOV. Jako první je potřeba zjistit, zda je možné, přenést pakety mezi jednotlivými OvS porty, ať už se jedná o reprezentátor nebo hlavní zařízení. Jelikož se OvS po startu chová jako rozvaděč (*hub*), je jednoduché otestovat, že se pakety dostanou z jednoho zařízení do druhého. K tomuto jsem použil příkazy `ndp-transmit` a `ndp-read` pro zaslání a získání testovacího paketu. Po tomto testování jsem zjistil, že komunikace funguje oběma směry pro všechny zařízení.

Dalším krokem je pak otestování nad reálným síťovým provozem. Tyto testy jsem provedl dva. V prvním testu jsem testoval komunikaci mezi dvěma virtuálními stroji, které byly připojeny na Open vSwitch. Na obrázku 3.6 je pro lepší představu vyobrazeno toto zapojení. Kromě již zmíněného nastavení, popsaného výše, je potřeba vytvořit virtuální stroje a napojit je na virtuální funkce. Pro vytvoření těchto virtuálních strojů jsem použil `libvirt` nástroje a jako operační systém virtuálních strojů jsem použil Fedora OS. Po vytvoření těchto virtuálních strojů jsem do nich připojil virtuální funkce, čímž jsem umožnil manipulaci s virtuální funkcí uvnitř VM. Následně jsem v obou virtuálních strojích nainstaloval Linux kernel ovladač *nfb* a uvedl jsem ho do režimu, ve kterém se dá použít jako klasické síťové rozhraní. Pak už stačilo správně nastavit IPv4 adresu a síťové přípojení pomocí příkazu `ifconfig`. Po tomto nastavení pak už zbývalo jen otestovat komunikaci. Tu jsem testoval pomocí příkazů `ssh` a `scp`, kde se cílový virtuální stroj identifikoval IPv4 adresou, která byla výše přidělena pomocí příkazu `ifconfig`. Pro ujištění, že pakety tečou

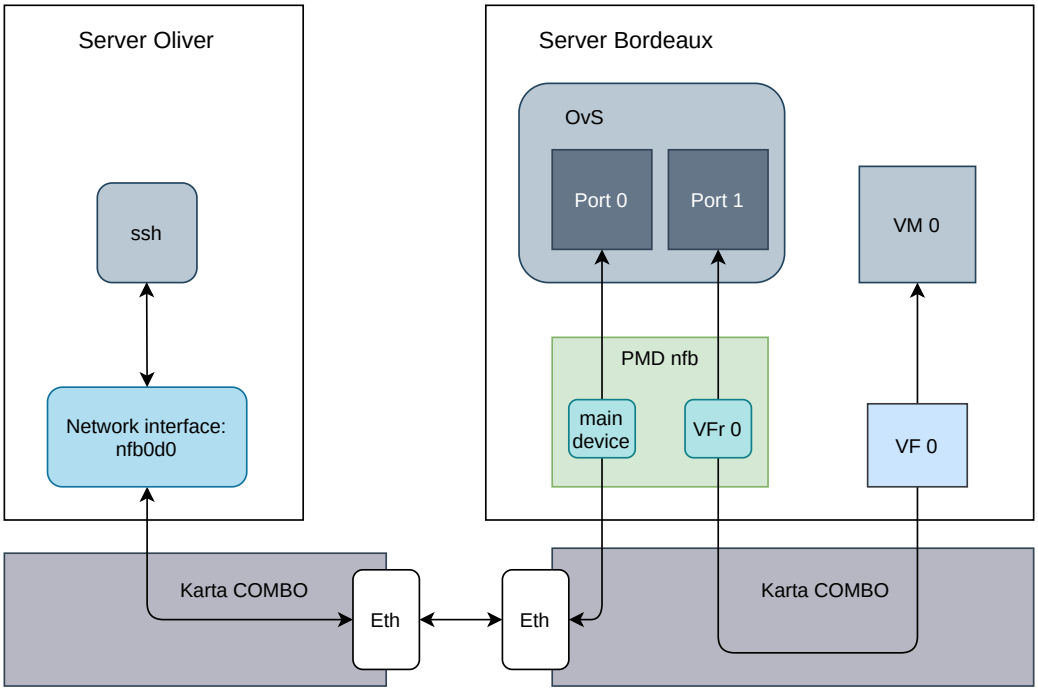


Obrázek 3.6: OvS běžící nad kartou COMBO se dvěma virtuálními stroji

správnou cestou, jsem sledoval počty přijatých a odeslaných paketů a bytů pomocí příkazu `ifconfig` ve virtuálních strojích a pomocí příkazu `ovs-ofctl` jsem sledoval počty přijatých a odeslaných paketů na serveru Bordeaux. Tento test dopadl pozitivně a ukázal, že přes toto SR-IOV rozhraní je možné posílat síťový provoz.

I když se po prvním testu může zdát rozhraní jako plně funkční, pořád nebyla otestována síťová komunikace, virtuálního stroje a počítačem mimo OvS. Proto se druhý test zabýval komunikací mezi virtuálním strojem a serverem, který byl připojen pomocí Ethernetového portu, dostupného na kartě. Tady jsem využil server Oliver, na kterém byla připojena připojena COMBO karta. Tato karta byla propojena pomocí Ethernetového portu s kartou na serveru Bordeaux, což je pro lepší představu vyobrazeno na obrázku 3.7. Nastavování virtuálního stroje probíhalo stejně jako ve výše zmíněném testování. Na serveru Oliver jsem nastavil Linux kernel ovladač *nfb* stejně jako na virtuálních strojích a pak jsem zde obdobně použil příkaz `ifconfig` pro konfiguraci síťového rozhraní. Po tomto nastavení jsem zase vyzkoušel síťové připojení pomocí příkazů `scp` a `ssh`. Tato komunikace byla taktéž monitorována pomocí příkazů `ifconfig` a `ovs-ofctl`. I tento test dopadl pozitivně, což prokazuje funkčnost podpory SR-IOV v PMD *nfb*.

V další části testování jsem se zaměřil na RTE flow podporu v ovladači. Tuto podporu jsem testoval jak pomocí testovací aplikace, tak pomocí OvS. V rámci tohoto testování jsem také implementoval funkci `print_p4_data()`, která při offloadu jakéhokoliv pravidla vypíše všechny parametry a klíče daného pravidla. Toto pak dost zjednodušuje testování, zda offload pravidla byl proveden a také zjišťování jaké hodnoty byly do P4 pravidla vloženy. Bohužel jsem ale narazil při testování na chybu vkládání P4 pravidla do karty. Tato chyba se projevuje tím, že při prvním běhu aplikace se P4 pravidla na pakety neaplikují. V dalších běžích aplikace pak P4 pravidla, která byla nahraná v posledním běhu, zůstávají v pozmě-

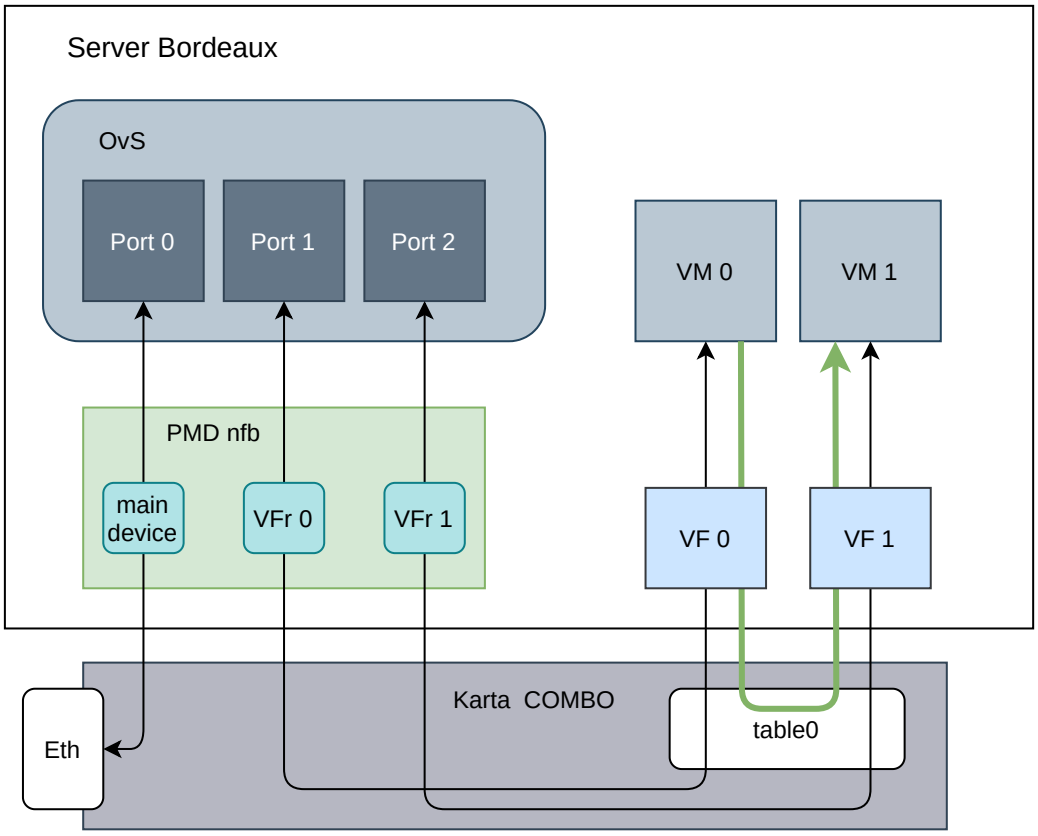


Obrázek 3.7: OvS nad kartou COMBO, která je propojená s jiným počítačem

něné podobě nahraná v kartě i přes volání funkcí pro uvolnění těchto pravidel a pro reset P4 zařízení. Tuto chybu jsem pak z části vyřešil, nicméně kvůli této chybě a jejímu řešení je možné vkládat pouze jedno RTE flow pravidlo do karty a inicializace P4 zařízení je nutná provádět již před zapnutím RTE flow aplikace.

V první části testování RTE flow podpory jsem použil testovací DPDK aplikaci, která zde již byla zmíněna. Aplikaci jsem upravil tak, aby byla schopná vytvářet RTE flow pravidla. Tato úprava spočívala pouze v definici právě testovaného pravidla a poté zavolání funkce `rte_flow_create`. Při testování jsem po zapnutí aplikace, což způsobilo nahrání právě testovaného pravidla, poslal paket do karty pomocí příkazu `ndp-transmit`. Následně jsem postupoval podle druhu nahraného pravidla. Pokud se jednalo o pravidla, která modifikovala části paketů, nahrál jsem výstupní paket pomocí příkazu `ndp-read` a pak jsem ho porovnal se vstupním paketem. Tyto pakety byly v uchovávány ve formátu *pcap* a pro jejich analýzu jsem použil programy `hexdump` a `wireshark`. Pokud se nejednalo o pravidla, které upravovala části paketu, postupoval jsem individuálně podle pravidla. V případě akce DROP jsem si ověřil, že vyslaný paket nebyl přijat na žádném rozhraní. V případě akce COUNT jsem použil funkce `rte_flow_query()`, ze které jsem vyčítal hodnotu z čítače. Tuto hodnotu jsem následně porovnával s počtem paketů poslaných do karty, které vyhovovali požadavkům *matching pattern*. V případě akce PORT_ID jsem sledoval komunikaci mezi jednotlivými reprezentátory a virtuálními funkcemi, abych rozpoznal, zda jsou pakety správně přesměrovány. Všechna tato pravidla uspěla v testování na této testovací DPDK aplikaci.

Další část testování RTE flow je zaměřena na Open vSwitch. V rámci tohoto testování jsem musel upravit `setup_ovs.sh` skript tak, aby umožnil OvS harware offload. Zde ale nastává problém, že Open vSwitch sám rozhoduje o tom, jaké pravidlo bude offloadovat do hardware a jaké bude nadále řešit na softwarové úrovni. Toto značně činnost testování



Obrázek 3.8: OvS nad kartou COMBO, s použitím akce PORT_ID

ztěžuje. Při běhu OvS se mi podařilo zachytit žádost o offload pravidla s akcemi COUNT a PORT_ID. Toto pravidlo bylo následně offloadováno, což je zobrazeno na obrázku 3.8. Akce PORT_ID je v OvS používaná pro přímé přeposílání paketů z jednoho virtuálního stroje do druhého, což znamená, že je provoz odkloněn mimo Open vSwitch. Proto jsem funkčnost této akce ověřil sledováním, zda přenášené pakety a byty prochází přes OvS pomocí příkazu `ovs-ofclt`. Zde se ukázalo, že akce PORT_ID v rámci OvS funguje a úspěšně odklání provoz.

Kapitola 4

Závěr

Cílem této práce bylo v první řadě seznámení s knihovnou DPDK, klasifikačním rozhraním RTE flow, jazykem P4, virtuálním přepínačem Open vSwitch a jeho možnostmi pro přesun klasifikačních pravidel na úroveň chytré síťové karty. V dalším kroku bylo nutné se seznámit s platformou NDK, P4 překladačem, který existuje v rámci této platformy a nakonec DPDK PMD ovladačem, vyvíjeným v rámci sdružení CESNET. Všechny tyto technologie jsou popsány v rámci teoretické části.

Dalším krokem pak bylo navrhnout vhodný způsob akcelerace virtuálního přepínače Open vSwitch s využitím platformy NDK, technologie SR-IOV a jazyka P4. Po tomto kroku následuje implementace navrženého řešení a ověření jeho funkčnosti na dostupném hardware. Toto je obsaženo v praktické části této bakalářské práce. Zde se ukazuje, že se mi povedlo implementovat a otestovat rozšíření DPDK PMD o technologii SR-IOV. Dále jsem navrhl a částečně implementoval v DPDK PMD podporu klasifikačního rozhraní RTE flow. Tato podpora byla implementovaná pouze částečně, jelikož zde pořád zbývá vyřešit již zmíněnou chybu a tím zajistit nahrání většího počtu RTE flow pravidel do karty.

Kromě již zmíněné opravy v RTE flow bych ještě rád uvedl několik možností, jak pokračovat v budoucnu a jak lze zlepšit současné řešení. Jako první by pak bylo vhodné zaměřit se na podporu RTE flow akce RRS v síťové kartě, jelikož tuto akci se OvS snaží aktivně offloadovat. Dále bych zmínil, že počet vložených pravidel do karty je značně omezen a to z důvodu použití P4 klíčů typu *ternary*. Tento problém se dá řešit přidáním externích pamětí, které umožní zvětšit kapacitu přidaných pravidel. Dalším problémem pak je cena a dostupnost karet COMBO. To je hlavní nevýhoda tohoto řešení v praxi. Tento problém se dá řešit přechodem na jinou architekturu. Zde je podstatné, aby nová karta disponovala FPGA čipem. Toto splňuje například karta N3000 od společnosti Intel, která v budoucnu může být využita pro tento přechod.

Literatura

[1] *Cards* [online]. [cit. 2020-27-04]. Dostupné z: https://www.liberouter.org/technologies/cards/.

[2] *Basic Forwarding Sample Application* [online]. 2020 [cit. 2020-13-05]. Dostupné z: https://doc.dpdk.org/guides/sample_app_ug/skeleton.html.

[3] *DPDK DOCUMENTATION* [online]. 2020 [cit. 2020-16-04]. Dostupné z: http://core.dpdk.org/doc/.

[4] *DPDK Physical Ports* [online]. 2020 [cit. 2020-13-05]. Dostupné z: http://docs.openvswitch.org/en/latest/topics/dpdk/phy/.

[5] *Generic flow API (rte_flow)* [online]. 2020 [cit. 2020-16-04]. Dostupné z: https://doc.dpdk.org/guides/prog_guide/rte_flow.html.

[6] *Open vSwitch Documentation* [online]. 2020 [cit. 2020-22-04]. Dostupné z: http://docs.openvswitch.org.

[7] *P4 Language and Related Specifications* [online]. 2020 [cit. 2020-23-04]. Dostupné z: https://p4.org/specs/.

[8] *P4 Netcope user guide*. Netcope, February 2020. [Online], Verze 1.0.5 (2020), [rev. 2020-2-24], [cit. 2020-4-22].

[9] Broadcom. *The birth of SmartNICs – offloading dataplane traffic to...software* [online]. [cit. 2020-05-05]. Dostupné z: https://www.slideshare.net/LF_OpenvSwitch/lfovs17the-birth-of-smartnics-offloading-dataplane-traffic-tosoftware.

[10] Broadcom. *Introduction to NetXtreme® Congestion Control (NCC)* [online]. [cit. 2020-05-05]. Dostupné z: https://docs.broadcom.com/doc/NCC-WP101.

[11] gorman, M. *Huge pages part 1 (Introdution)* [online]. 2010 [cit. 2020-15-04]. Dostupné z: https://lwn.net/Articles/374424/.

[12] Hat, R. *Red Hat's perspective on OVS HW Offload Status* [online]. [cit. 2020-04-05]. Dostupné z: https://www.slideshare.net/LF_OpenvSwitch/lfovs17enabling-hardware-acceleration-in-ovsdpdk-using-dpdk-framework.

[13] Lacerda Ruivo, T. P. P. de et al. *Efficient High-Performance Computing with, Infiniband Hardware Virtualization* [online]. 2014 [cit. 2020-21-4]. Dostupné z: http://datasys.cs.iit.edu/reports/2014_IIT_virtualization-fermicloud.pdf.

[14] Marvell. *Enabling Hardware Offload of OVS Control & Data plane using LiquidIO* [online]. [cit. 2020-05-05]. Dostupné z: https://www.slideshare.net/LF_OpenvSwitch/lfovs17enabling-hardware-offload-of-ovs-control-data-plane-using-liquidio.

[15] Marvell. *LiquidIO II 10/25GbE Adapter family* [online]. [cit. 2020-05-05]. Dostupné z: https://www.marvell.com/products/ethernet-adapters-and-controllers/liquidio-smart-nics/liquidio-ii-smart-nics.html.

[16] Mellanox. *ConnectX®-5 EN Card* [online]. [cit. 2020-05-05]. Dostupné z: https://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-5_EN_Card.pdf.

[17] Mellanox. *OVS Performance on Steroids - Hardware Acceleration Methodologies* [online]. [cit. 2020-05-05]. Dostupné z: https://www.slideshare.net/LF_OpenvSwitch/lfovs17ovs-performance-on-steroids-hardware-acceleration-methodologies.

[18] Napatech. *Napatech SmartNIC Solutions Virtual Switch Acceleration* [online]. [cit. 2020-05-05]. Dostupné z: https://www.napatech.com/support/resources/solution-descriptions/napatech-smartnic-solutions-virtual-switch-acceleration/.

[19] Netronome. *Agilio CX SmartNICs* [online]. [cit. 2020-05-05]. Dostupné z: https://www.netronome.com/products/agilio-cx/.

[20] Netronome. *OvS Hardware Offload with TC Flower* [online]. [cit. 2020-05-05]. Dostupné z: https://www.slideshare.net/LF_OpenvSwitch/lfovs17ovs-hardware-offload-with-tc-flower.