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 V DPDK

ACCELERATION OF OPEN VSWITCH IN DPDK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. DAVID VODÁK**
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 diplomové práce

Student: **Vodák David, Bc.**
Program: Informační technologie a umělá inteligence
Specializace: Vestavěné systémy
Název: **Akcelerace virtuálního přepínače Open vSwitch v DPDK**
Acceleration of Open vSwitch in DPDK
Kategorie: Počítačové sítě
Zadání:

1. Seznamte se s knihovnou DPDK, klasifikačním rozhraním RTE Flow, virtuálním přepínačem Open vSwitch (OvS) a jeho možnostmi pro přesun klasifikačních pravidel na úroveň chytré síťové karty.
2. Seznamte se s platformou pro chytré síťové karty PAC N3000 od společnosti Intel, jejím DPDK ovladačem a firmware pro akceleraci OvS vyvíjeným sdružením CESNET.
3. Navrhněte vhodný způsob akcelerace OvS přepínače (v DPDK) na platformě Intel PAC N3000.
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.

Při obhajobě semestrální části projektu 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: 18. května 2022
Datum schválení: 29. října 2021

Abstrakt

Virtuální přepínač je software, který připojuje virtuální stroje k síti, což z něj dělá nedílnou součást virtualizace na serverech. Nicméně při vyšších síťových rychlostech se stává neefektivní, jelikož všechny rámce přepíná softwarově. Tato práce se zabývá hardwarovou akcelerací virtuálního přepínače Open vSwitch. Akcelerační prototyp, který je cílem této práce, je založen na rozhraní RTE flow, standardu SR–IOV a kartě PAC N3000 od společnosti Intel. V rámci této diplomové práce byly popsány technologie potřebné pro akceleraci, poté byl vytvořen návrh akceleračního prototypu, ten byl následně implementován a otestován. Nakonec byla měřena propustnost a bylo zjištěno, že pravidla nahraná do hardware v rámci akceleračního prototypu zvyšují propustnost.

Abstract

Virtual switch is a software that connects virtual machines to the internet, which makes it a crucial part of virtualization on servers. Nevertheless, it can be rather ineffective when it comes to high speed traffic, since it switches all frames in the software. This thesis is about hardware acceleration of the virtual switch called Open vSwitch. The acceleration prototype, which is the goal of this thesis, is based on the RTE flow interface, the SR–IOV standard, and Intel PAC N3000 card. In the scope of this master's thesis, all necessary technologies were described and the acceleration prototype was designed, implemented, and tested. Results of executed measurements indicate increased throughput when rules of the acceleration prototype were offloaded to hardware.

Klíčová slova

DPDK, Open vSwitch, OvS, Intel PAC N3000, hardwarová akcelerace, SR–IOV, RTE flow, VFIO, FPGA

Keywords

DPDK, Open vSwitch, OvS, Intel PAC N3000, hardware acceleration, SR–IOV, RTE flow, VFIO, FPGA

Citace

VODÁK, David. *Akcelerace virtuálního přepínače Open vSwitch v DPDK*. Brno, 2022. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Martínek, Ph.D.

Akcelerace virtuálního přepínače Open vSwitch v DPDK

Prohlášení

Prohlašuji, že jsem tuto diplomovou 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
15. května 2022

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Tomáši Martínkovi, Ph.D. za ochotu ke konzultacím a přínosné rady. Dále bych rád poděkoval členům sdružení CESNET v aktivitě programovatelný hardware za odbornou pomoc a příjemné pracovní prostředí.

Obsah

Kapitola 1

Úvod

Virtualizace na serveru je v dnešní době hojně používaná a díky ní je možné rozdělit server na více virtuálních strojů. Tyto stroje mohou pracovat paralelně a na sobě nezávisle. Správce serveru může nastavit, jaké zdroje (CPU, paměť RAM) kterému virtuálnímu stroji přidělí. Díky tomu je provozovatel schopen poskytnout zákazníkovi přesně tolik zdrojů, kolik je potřeba, bez zbytečného plýtvání. Prakticky tedy může běžet několik virtuálních strojů na jednom fyzickém, z nichž každý má přidělen požadovaný počet zdrojů a operační systém, které si zákazníci vyžádali.

Pro připojení virtuálních strojů k síti se typicky využívá virtuální přepínač. Ten má své porty připojeny k jednotlivým virtuálním strojům a také alespoň k jedné síťové kartě. Po počáteční konfiguraci všech potřebných portů se virtuální přepínač věnuje přepínání rámců, čímž je umožněna komunikace mezi jednotlivými virtuálními stroji, popřípadě se vzdáleným počítačem, který je připojený k síti. Virtuální přepínač je software, a proto veškerou práci vykonává pomocí procesoru.

Softwarové zpracování rámců pomocí CPU je výpočetně náročné v rámci vyšších rychlostí. Vzhledem k zvyšujícím se nárokům na síťovou propustnost je možné se v dnešní době setkat s rychlostmi jako je 10, 50 nebo i 100 Gb/s. V takovém případě spotřebuje server hned několik jader CPU pouze na přepínání rámců. Tato jádra by mohla být použita jedním nebo i více virtuálními stroji, které by mohly navíc běžet na serveru.

Pro zpracování síťových toků o takto vysokých rychlostech je vhodné na místo CPU použít hardware. Zde se nabízí chytré síťové karty, které jsou schopny vykonávat určité úkony za virtuální přepínač. Tvůrci virtuálních přepínačů jsou si toho vědomi a rozšiřují je o nové rozhraní pro offload klasifikačních pravidel do hardware.

V této práci akceleruji virtuální přepínač Open vSwitch (OvS) v režimu DPDK. Zde bylo tvůrci pro tento účel vytvořeno rozhraní pro akceleraci pomocí RTE flow. Díky tomu je OvS umožněna realizace některých akcí nad rámci přímo v hardware, čímž je ušetřen výkon CPU.

Cílem této práce je implementovat RTE flow rozhraní pro kartu PAC N3000 od společnosti Intel. Ta disponuje čipem FPGA, jenž bude použit k poskytnutí podpory klasifikačních pravidel virtuálnímu přepínači OvS.

Tato práce se dělí na teoretickou a praktickou část. V první kapitole teoretické části je popsána virtualizace a virtualizační technologie 2.1, což zahrnuje hypervizor Qemu a virtuální přepínač Open vSwitch. V další kapitole jsou popsány problémy virtualnizačních technologií a jejich řešení 2.2, ty zahrnují ovladač VFIO, standard SR–IOV a možnosti hardwarové akcelerace OvS. Další kapitola se věnuje kartě PAC N3000 od společnosti Intel 2.3, zde jsou také popsány čipy Arria 10 a XL710, které jsou součástí této karty. Dále tato

kapitola zahrnuje OPAE framework, který se používá pro práci s kartou. V další kapitole je popsán firmware pro akceleraci OvS 2.4, který je vyvíjen sdružením CESNET a dá se použít i v rámci PAC N3000. Poslední kapitola teoretické části je věnována frameworku DPDK 2.5, kde se popisuje akcelerační rozhraní RTE flow, ovladač pro XL710 `i40e` a ovladač pro PAC N3000 `ipn3ke`. První kapitola praktické části se zabývá návrhem a implementací akceleračního prototypu OvS 3.1. V následujících kapitolách je popsáno testování tohoto prototypu 3.2 a měření jeho propustnosti 3.3.

Kapitola 2

Teoretická část

V teoretické části jsou popsány virtualizační technologie, jejich úzká hrdla a možnosti jejich řešení. Následně jsou zde popsány technologie, které jsou použity v praktické části pro akceleraci virtuálního přepínače Open vSwitch. Mezi tyto technologie patří framework pro rychlé zpracování paketů DPDK, chytrá síťová karta PAC N3000 od společnosti Intel a firmware pro akceleraci OvS vyvíjený sdružením CESNET.

2.1 Virtualizace a virtualizační technologie

Virtuální stroj je počítač, který nepracuje přímo s hardwarem hostitelského stroje. Jeho zdroje mu přiděluje hypervisor, což je software, který se stará o běh virtuálních strojů. Díky tomu je možné, aby na jednom fyzickém stroji běželo více virtuálních strojů současně [41].

Hypervisory se dělí na dva typy, kde typ jedna používá přímo hostitelský hardware a typ dva má přiděleny zdroje od operačního systému jako normální program. Hypervisory typu 1, též zvané nativní, jsou výkonnější, jelikož fyzické zdroje přerozdělují přímo. Z tohoto důvodu jsou také mnohem používanější než hypervisory typu 2. Mezi nativní hypervisory patří například Hyper-V od společnosti Microsoft nebo VMware vSphere. Ty jsou nainstalovány přímo na fyzickém hardware jako výchozí systém. Dalším nativním hypervisorem je KVM (Kernel-based Virtual Machine), což je specifická virtualizační technologie, která je součástí linuxového jádra od verze 2.6.20. Díky této technologii je možné přímo předat hardwarové zdroje, i když na hostitelském stroji běží operační systém. Hypervisory typu 2 jsou známé i jako hostované, jelikož běží jako programy v rámci hostitelského operačního systému. Mezi takové patří například Virtualbox od firmy Oracle, VMware Workstation nebo QEMU [21, 42].

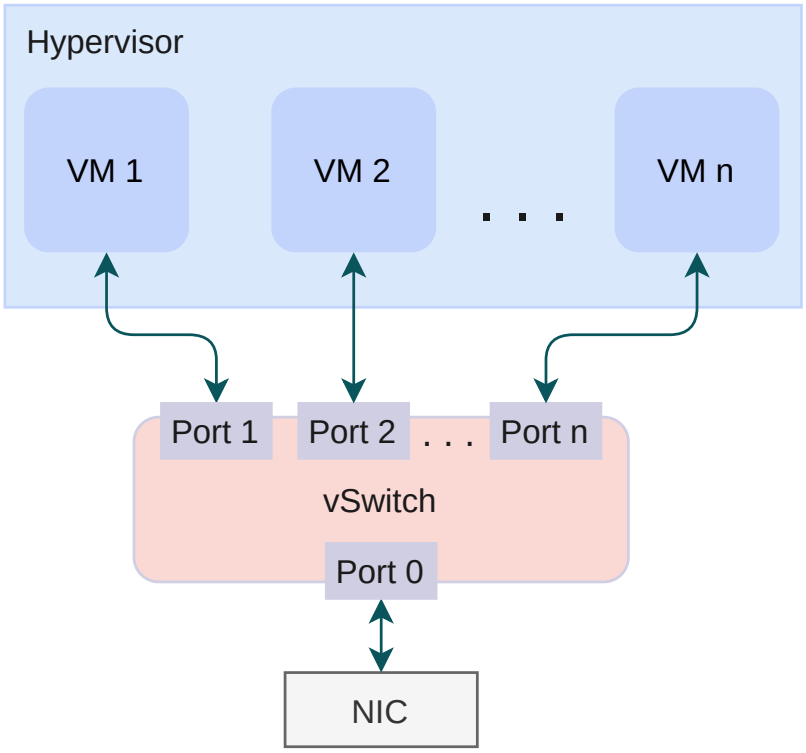
Pro používání virtuálních strojů je nutné v daném hypervisoru nastavit přidělení potřebných zdrojů. Mezi ty hlavní patří počet logických jader procesoru, množství operační paměti a paměti pevného disku. Dalším možným nastavením je přidělení jiných hadrwarových prostředků, jako je například síťová nebo grafická karta. Po tomto základním nastavení je možné stroj spustit a ten pak již funguje jako fyzický počítač, který ovšem běží jen dokud mu to hypervisor umožní [33].

Virtuální stroje je vhodné používat kvůli optimalizaci zdrojů. V dnešní době mají servery vetšinou více zdrojů než kolik si zákazník žádá, díky virtualizaci je možné tyto zdroje rozdělit mezi více zákazníků, čímž se lépe využijí. Častou praktikou je také pouštění pouze jedné aplikace na daném počítači. Zde se opět vyplatí užití virtuálních strojů, jelikož jedna

aplikace typicky nevytíží celý fyzický server. Dalším důvodem proč používat virtualizaci je rychlejší údržba. Například nasazení je mnohem jednodušší, jelikož stačí pouze najít vhodný hostitelský server, kde je dostatek zdrojů pro daný virtuální stroj. Pokud se vyskytne chyba na hostitelském serveru, stačí, aby byl virtuální stroj přenesen na jiný fyzický server a může být nadále používán. Dalším způsobem jak minimalizovat ztráty při chybách je spuštění více virtuálních strojů se stejnou aplikací, které jsou umístěny na odlišných fyzických serverech. Pokud pak nastane chyba na jednom z hostitelských serverů, aplikace i nadále běží na ostatních virtuálních strojích. Další výhodou je možnost spouštění programů, které jsou určeny pro starý hardware a nejsou přenositelné. Díky virtuálním strojům je možné tyto programy i nadále používat na moderních serverech [35].

Virtuální stroje je možné připojit k síti obecně dvěma způsoby. Prvním způsobem je přidělení síťové karty virtuálnímu stroji. Tento přístup je velmi efektivní, co se výkonnosti týče, nicméně maximální počet virtuálních strojů je omezen množstvím síťových karet, které jsou na serveru instalovány. Z tohoto důvodu není tento přístup příliš praktický a v praxi se nepoužívá. Další možností je použití virtuálního přepínače [45].

Virtuální přepínač (anglicky virtual switch, zkráceně vSwitch) je software, který je přepíná rámce mezi síťovou kartou a virtuálními stroji. Tím je zajištěno připojení virtuálních strojů k síti. Pro připojení virtuálního přepínače k virtuálním strojům je nutné, aby hypervisor emuloval síťovou kartu, která je následně připojena k virtuálnímu přepínači. Na obrázku 2.1 je možné vidět obecnou architekturu virtuálního přepínače [32].



Obrázek 2.1: Obecné zapojení virtuálního přepínače

2.1.1 Qemu

QEMU (Quick Emulator) je hypervisor a emulátor, který je schopen vytvářet virtuální stroje, starat se o jejich běh nebo spouštět procesy, které byly přeloženy pro jiné CPU než je CPU hostitelského stroje. QEMU je schopné emulovat pro virtuální stroj CPU, operační paměť i spoustu dalších zařízení. V takovémto případě se chová jako hypervisor typu 2. Nicméně QEMU je schopné kooperovat i s jinými hypervisory, velmi často se pak používá kooperace s KVM. V takovém případě používá virtuální stroj CPU napřímo bez nutnosti emulace [5, 43].

Uživatelské rozhraní QEMU je pouze v příkazové řádce v podobě jednotlivých utilit. Pro tvorbu, konvertování nebo modifikaci diskových jednotek virtuálních strojů existuje příkaz `qemu-img`. Hlavní příkaz pro spouštění virtuálního stroje se jmenuje `qemu`, popřípadě `qemu-var`, kde `var` je specifická varianta QEMU. Tento příkaz obsahuje velké množství argumentů, ve kterých lze přidat nějaké reálné nebo emulované zařízení a spoustu dalšího. Pro ulehčení práce s QEMU je možné využít knihovny `libvirt`, kterou používají programy poskytující více uživatelsky přívětivé rozhraní. Takových programů je mnoho, například `virsh`, který funguje pouze na příkazové řádce. Grafické uživatelské rozhraní poskytuje například `virt-manager` nebo GNOME boxes [5, 44].

QEMU je projekt s otevřeným vývojem. To znamená, že zdrojové kódy jsou veřejně k dispozici a uživatel si tak sám může editovat zdrojové kódy, přeložit je a mít tak svoji vlastní verzi programu. K tomuto projektu také existuje emailová konference, kde vývojáři řeší různé problematiky, které se týkají vývoje. Probírají se zde také návrhy na rozšíření, díky čemuž se uživatel může k novým plánovaným změnám dostat dříve než se dostanou do oficiální verze programu [5].

2.1.2 Open vSwitch

Open vSwitch (zkráceně OvS) je, jak už název napovídá, virtuální přepínač s otevřeným vývojem. Tento přepínač podporuje mnoho hypervisorů, jako je například Virtualbox, nebo KVM. Podporuje operační systémy Linux, NetBSD a FreeBSD. Tento virtuální přepínač je schopen běžet v Linux kernel režimu nebo i v DPDK režimu (též nazýváno OvS–DPDK). V prvním případě používá pro připojení k síti standardní linuxové síťové rozhraní a v druhém jsou použita DPDK zařízení [4].

Open vSwitch podporuje spoustu vlastností a funkcionalit, jako je například standardní 802.1Q VLAN model. V rámci tohoto modelu je možné označit porty OvS jako trunk nebo access a tak zajistit přidání či odebrání VLAN hlaviček. Dále Open vSwitch podporuje standard OpenFlow, díky čemuž je možné vkládat do OvS pravidla pomocí OpenFlow kontroléru. Další zajímavou vlastností je Linux kernel modul, který je schopen zvýšit výkon klasické kernel varianty OvS [4].

OvS se vnitřně dělí na `bridge`, kde každý `bridge` funguje jako přepínač, takže je možné, aby v rámci jednoho serveru běželo více OvS přepínačů (`bridge`). Každý `bridge` přepíná pakety mezi porty, ty mohou být připojeny buď k fyzickým nebo virtuálním zařízením [4].

Jak již bylo zmíněno výše, OvS může fungovat ve dvou režimech. Linux kernel režim je sice pomalejší než používání DPDK, nicméně je stabilnější a podporuje více funkcionalit, které v DPDK režimu ještě nebyly naimplementovány. Naproti tomu DPDK režim se jeví jako rychlejší alternativa, která je ovšem méně stabilní. DPDK varianta je ale v poslední době značně rozvíjena a předpokládá se, že v budoucnu bude více stabilní [4].

OvS zahrnuje příkazy pro spouštění, konfiguraci a spoustu dalšího. Příkaz, který implementuje samotný virtuální přepínač, se jmenuje `ovs-vswitchd` a při jeho spuštění

se vytvoří `brigde` a porty, které jsou uvedené v OvS databázi. Tuto databázi implementuje příkaz `ovsdb-server`, který musí být spuštěný ještě před `ovs-vswitchd`. Jedná se o jednoduchou databázi, ze které `ovs-vswitchd` vyčítá svou konfiguraci. Uživatel do ní může zapisovat, popřípadě se na ni dotazovat pomocí příkazu `ovs-vsctl`. Dalším důležitým nástrojem je `ovs-ofctl`, který slouží pro správu a dotazování se OpenFlow přepínačů a kontrolérů. Pomocí něj se dá například zjistit, kolik rámců již prošlo přes daný `bridge` [4, 22].

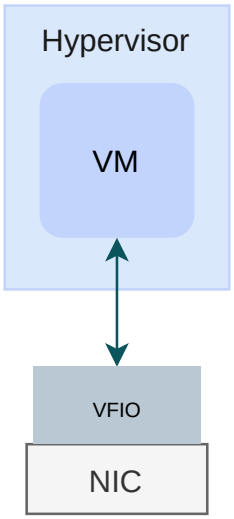
2.2 Problémy virtualizačních technologií a jejich řešení

I když je virtualizace v praxi hojně využívaná, má bohužel i své nevýhody. Ty se projevují zejména u řešení, kde je zapotřebí vysoké síťové propustnosti. První z těchto nevýhod je softwarová emulace síťových karet ve virtuálních strojích. Pro tuto emulaci se typicky používá VIRTIO frameworku, pomocí kterého je možné vytvořit virtuální síťovou kartu, která ovšem spotřebovává výkon hostitelského serveru. Tato spotřeba je pak závislá na síťové rychlosti [30, 40].

Dalším problémem je činnost samotného virtuálního přepínače. Přepínání rámců zatěžuje procesor hostitelského serveru a při vysokých rychlostech může plně vytížit hned několik CPU. Pro Open vSwitch v DPDK režimu (který je považován za rychlejší variantu) bylo naměřeno, že zpracování síťového provozu o rychlosti 4 milionů paketů za sekundu zatíží plně jedno jádro procesoru. V dnešní době se mohou vyskytnout i vyšší síťové rychlosti a v takovém případě si provoz OvS vyžádá hned několik CPU [38].

2.2.1 VFIO

Virtual Function I/O, zkráceně VFIO, je ovladač v linuxovém jádře, který je také framework pro zabezpečené předávání zařízení do userspace. Tento ovladač tedy poskytuje možnost ovládat zařízení v userspace aplikaci na velmi nízké úrovni, bez nutnosti použití standardních linuxových ovladačů [2].



Obrázek 2.2: Připojení síťové karty do virtuálního stroje pomocí VFIO

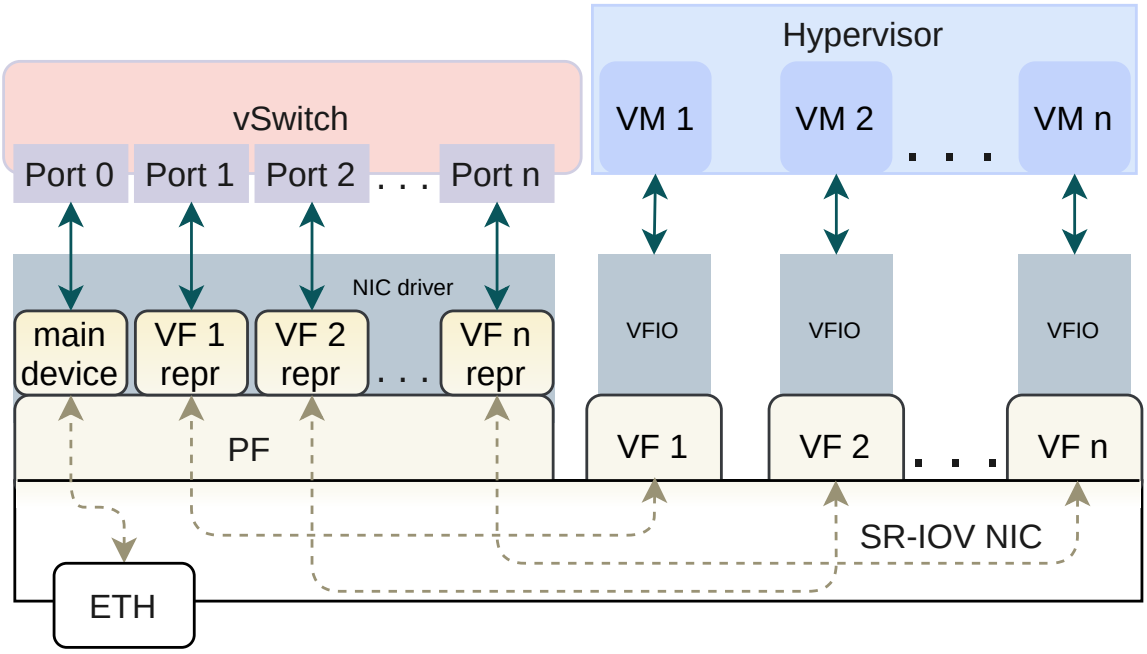
VFIO se dá použít pro řadu různých zařízení, nicméně z hlediska této práce je důležité jeho použití u síťových karet. Tento framework lze využít pro předání zařízení do virtuálního stroje. Hypervisor je díky tomuto ovladači schopen poskytnout rozhraní pro práci se síťovou kartou, které si ovšem nežádá takovou režii, jako tomu bylo u VIRTIO. Nevýhodou tohoto přístupu je ale fakt, že VFIO zařízení je možné používat pouze jednou aplikací. To znamená, že jedna síťová karta je schopna připojit pomocí této techniky pouze jeden virtuální stroj. Připojení virtuálního stroje k síti pomocí VFIO je znázorvněno na obrázku 2.2 [2, 37].

Tento ovladač se také často používá ve frameworku DPDK, jehož Poll Mode Drivery běží v userspace. VFIO tedy nabízí bezpečnou variantu jak používat zařízení v userspace například oproti staršímu, méně robustnějšímu ovladači UIO [1].

2.2.2 SR–IOV

Používání VFIO ve virtualizaci sice má své výhody, nicméně fakt, že pro připojení virtuálního stroje k síti je potřeba jedna fyzická síťová karta z něj dělá řešení, které není vhodné pro komerční nasazení. To vše ale mění standard SR–IOV.

Single Root Input/Output Virtualization, zkráceně SR–IOV, je standard, který umožňuje, aby PCI zařízení byla reprezentována více hardwarovými funkcemi. Existuje zde koncept fyzických a virtuálních funkcí. Fyzická funkce (Physical Function, zkráceně PF) je plnohodnotné PCI zařízení, které může řídit a konfigurovat PCI zařízení. Samozřejmě může také provádět Input/Output operace. Oproti tomu virtuální funkce (Virtual Function, zkráceně VF) má velmi omezené možnosti řízení a konfigurace zařízení. Prakticky slouží jen pro Input/Output operace a jejich počet je limitován I/O kanály (I/O streams) [29].

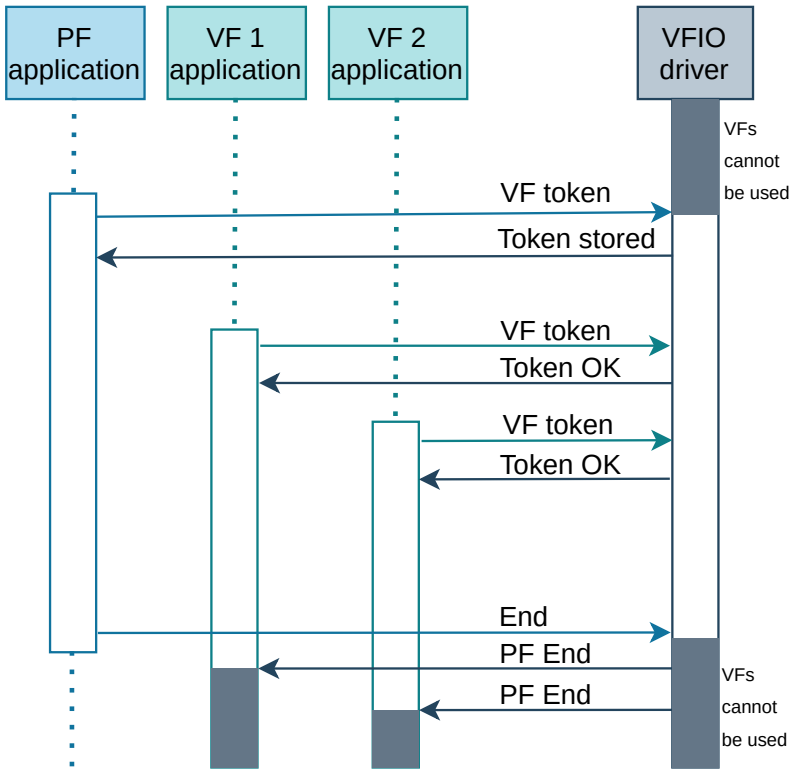


Obrázek 2.3: Zapojení virtuálního přepínače pomocí SR–IOV

Tento standard se dá použít pro síťové karty a tím je rozdělit na více hardwarových funkcí. Virtuální funkce pak mohou být předány skrze VFIO do virtuálních strojů a fyzická funkce je použita virtuálním přepínačem. Tímto způsobem je možné připojit více virtuálních strojů k síti pomocí jedné síťové karty za použití VFIO. Pro realizaci tohoto připojení je

ale nutná jak hadrwarová, tak softwarová podpora, kterou musí zajistit výrobce dané karty. Pokud existuje hardwarová podpora pro SR–IOV a kernel ovladač dané karty je schopen vytvářet virtuální funkce, je možné je vytvořit a pomocí VFIO připojit k virtuálním strojům [39].

Takto nastavené virtuální stroje pak lze přes síťovou kartu připojit k virtuálnímu přepínači. Virtuální funkce jsou namapovány na své reprezentátory (zařízení uvnitř fyzické funkce). Fyzickou funkci využívá virtuální přepínač, který si připojí reprezentátory ke svým portům, čímž je napojen na virtuální stroje. Dále virtuální přepínač využívá hlavní zařízení fyzické funkce pro připojení k síti. Toto zapojení je znázorněno na obrázku 2.3 [34].



Obrázek 2.4: Běh aplikací používající VFIO SR–IOV hardwarové funkce

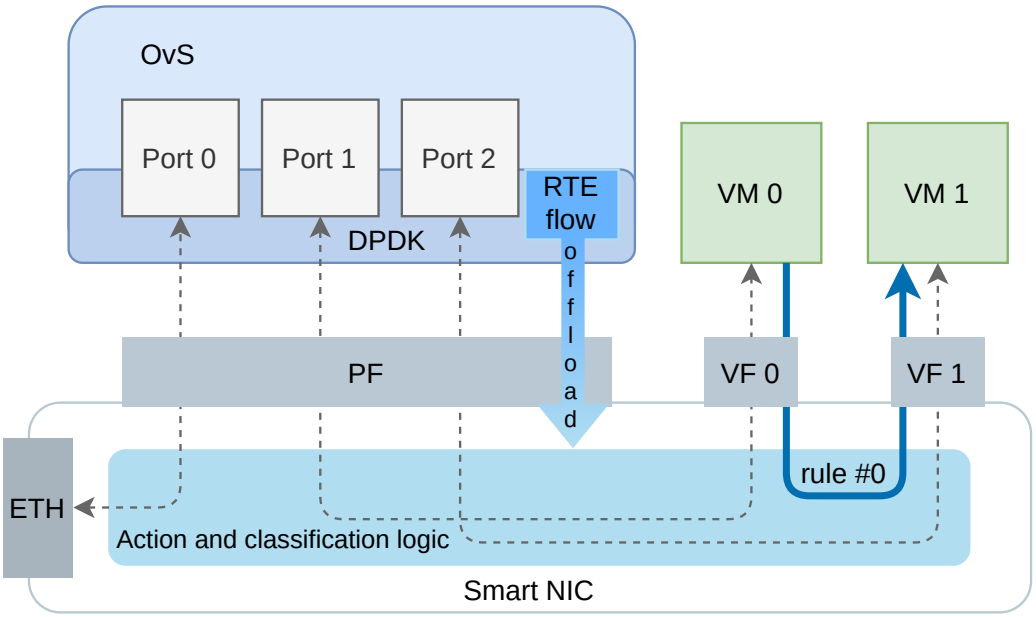
SR–IOV standard je nově podporován i v ovladači VFIO (linux kernel verze 5.7 a výše). To znamená, že i fyzická funkce může být ovládána z userspace aplikace bez nutnosti použití kernel ovladačů daných karet. Tento princip se dá využít zejména ve frameworku DPDK, který benefituje z používání VFIO. Jelikož VFIO klade důraz na bezpečnost, je zde přidán speciální bezpečnostní mechanismus, který zaručuje, že všechny hardwarové funkce používá stejný uživatel. Při spuštění aplikace, která používá fyzickou funkci, je nutné předat VFIO ovladači takzvaný VF token. Tento token má podobu UUID, což je identifikátor, který je možné vygenerovat například pomocí linuxového příkazu `uuid`. Po inicializaci PF aplikace, která zahrnuje i předání VF tokenu, je možné spustit aplikaci, která používá virtuální funkci. Zde je nutné předat stejný VF token, který dostal VFIO ovladač od PF aplikace. Tímto způsobem pak lze sputit i další aplikace, které používají další virtuální funkce. Po konci běhu PF aplikace skončí platnost VF tokenu a VF aplikace nadále nemohou používat virtuální funkce. Tento princip je znázorněn na obrázku 2.4. Jelikož tento mechanizmus vyžaduje předávání VF tokenu, znamená to úpravu všech aplikací, které VFIO s SR–IOV

používají. V DPDK již existuje tato podpora, která umožňuje DPDK aplikacím, aby používali buď jednu fyzickou nebo několik virtuálních funkcí (používání PF společně s VF alespoň prozatím není možné). Pro hypervizor a emulátor QEMU byl již vytvořen patch, který je možné použít pro připojení VFIO virtuálních funkcí do virtuálních strojů [46, 1].

2.2.3 Možnosti HW akcelerace OvS

Samotné použití SR–IOV s virtuálními funkcemi, které se přes VFIO napojí na virtuální stroje, je sice kompletně funkční, nicméně je zde pořád prostor pro akceleraci. Přepínání rámců totiž pořád vykonává virtuální přepínač v rámci software a hardware síťové karty pouze mapuje virtuální stroje na porty virtuálního přepínače.

Virtuální přepínače jako je například Open vSwitch nabízí možnost akcelerace pomocí offloadu klasifikačních pravidel do síťové karty. Tato pravidla jsou následně vykonávána v hardware a tím pádem nejen zrychlují činnost virtuálního přepínače, ale také snižují spotřebu CPU na hostitelském serveru.



Obrázek 2.5: Akcelerovaná verze OvS–DPDK

Open vSwitch podporuje hardwarový offload jak v linux kernel, tak v DPDK režimu. V rámci tradiční linux kernel varianty je použito rozhraní TC flower. Tato práce se ale věnuje akceleraci OvS v DPDK, a proto jsou zde popsány možnosti hardwarového offloadu v OvS–DPDK. Pro tuto variantu OvS je použito rozhraní RTE flow, které je podrobně popsáno ve své vlastní podkapitole 2.5.1. RTE flow pravidla se dělí na dvě nejdůležitější části. První z nich je vzor (`matching pattern`), podle kterého se pakety srovnávají a zjišťuje se, zda nedochází ke shodě. Druhou částí jsou akce (`actions`), které se mají nad paketem provést, pokud ke shodě paketu se vzorem dojde. Typickým pravidlem, které OvS offloaduje do hardware, je přesměrování. Vzor zde popisuje daný síťový tok, který míří přes OvS do virtuálního stroje. Důležitou akcí v tomto pravidle je pak `Output`, která je též nazývaná `PORT_ID` v RTE flow terminologii. Tato akce zapříčiní, že všechny pakety, které se shodují se vzorem, jsou přesměrovány přímo do virtuálního stroje a tím pádem se do OvS vůbec

nedostanou. To znamená, že OvS na přepínání tohoto síťového toku nespotřebuje vůbec žádný výkon CPU, jelikož bylo vše vyřešeno již v hardware [4, 9].

Na obrázku 2.5 je možné vidět akcelerovanou verzi OvS–DPDK, která používá RTE flow offloadu. Dále je zde znázorněno pravidlo `rule #0`, které bylo nahráno do chytré síťové karty a způsobuje přesměrování paketů v VM0 do VM1.

Podporované položky paketu, pomocí kterých OvS může definovat vzor, jsou Ethernet, VLAN, IPv4, IPv6, TCP, UDP, SCTP a ICMP. Mezi podporované akce, které se mohou pro offload použít, patří již zmíněná akce Output, která má na svědomí přesměrování paketů. Další akcí je Drop, což je akce, která způsobí zahození paketu. Následně zde existují akce pro editaci hlavičkek protokolů Ethernet, IPv4, IPv6, TCP, UDP. Poslední podporované akce tvoří VLAN Push a Pop pro offloadování pravidel, které realizují VLAN model [4].

2.3 Intel PAC N3000

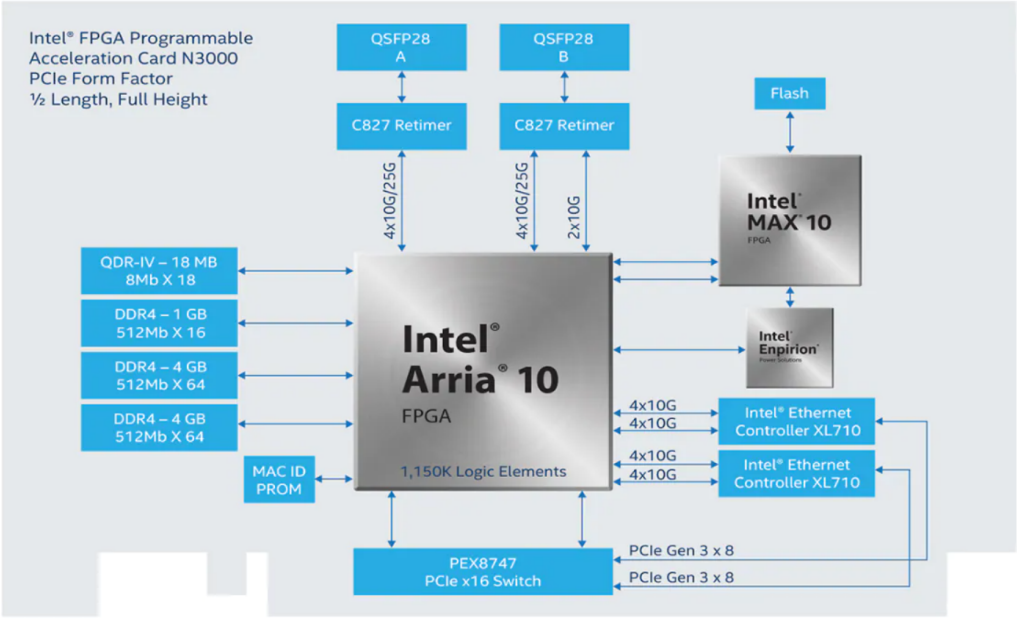
Karta PAC N3000 od společnosti Intel je opatřena čipem FPGA Arria 10 s označením GT 1150, dále disponuje síťovými adaptéry XL710, které mají podporu SR-IOV. To z ní dělá vhodného kandidáta pro akceleraci OvS, jelikož je schopna poskytnout mapování virtuálních strojů na porty OvS a zároveň může být v FPGA čipu implementována podpora pro offload klasfikačních pravidel do hardware. Další nespornou výhodou této karty je použití rozšířených XL710 adaptérů, jejichž ovladač je součástí linuxového jádra. Díky tomu není potřeba instalovat specifický ovladač ve virtuálních strojích, což usnadňuje nasazení této karty v data centru [3].

Obrázek 2.6: Karta Intel PAC N3000 [8]

Jádro této karty tvoří již zmíněný čip FPGA Arria 10, ten je napojen na dva ASIC čipy Intel XL710, které jsou připojeny ke sběrnici PCIe. Jelikož je čip XL710 plnohodnotná síťová karta, FPGA designer již nemusí implementovat DMA přenosy z karty do paměti hostitelského stroje. Na druhé straně je FPGA Arria 10 připojeno ke dvěma C827 Retimerům, které jsou napojeny na QSFP28 transceivery. Tuto síťovou architekturu ještě doplňuje FPGA MAX 10, které má na starosti řízení a monitorování desky. Pomocí něj se dají zjistit například hodnoty teploty nebo napětí na kartě. Tato karta je k dostání ve třech hardwarových konfiguracích. První z nich je 8 × 10GbE a nabízí na každém čipu XL710 a každém transceiveru čtyři kanály o rychlosti 10 Gb/s. Druhou konfigurací je 2 × 2 × 25GbE, která nabízí na každém XL710 dva kanály o rychlosti 40 Gb/s a každém transcevieru dva kanály

o rychlosti 25 Gb/s. Poslední konfigurací je 4 × 25GbE, v takovém případě je použit pouze jeden transceiver se čtyřmi kanály o rychlosti 25 Gb/s, čipy XL710 mají jako v předchozí konfiguraci oba dva kanály o rychlosti 25 Gb/s [3].

Hlavní část této karty je již zmíněné FPGA Arria 10 s označením GT 1150 od společnosti Intel, které obsahuje 1,15 milionu logických elementů, 1518 DSP bloků a 65,7 Mb paměti na čipu. Dále tento čip obsahuje 72 transceiverů o rychlosti 25.78 Gb/s. Ty se dají v rámci PAC N3000 použít buď pro připojení k XL710 nebo k QSFP28 transceiverům. Toto FPGA nepatří mezi nejvýkonnější, které společnost Intel nabízí, nicméně, je pořád schopné poskytnout dostatečný výkon a jeho cena dělá z karty PAC N3000 komerčně dostupnou variantu [23, 24].



Obrázek 2.7: Blokové schéma PAC N3000 [8]

2.3.1 OPAE framework

Open Acceleration Engine, zkráceně OPAE je framework pro práci s FPGA od společnosti Intel, který je napsán obecně pro FPGA čipy sloužící různým účelům. Je používán pro FPGA na kartě PAC N3000, ale i například pro procesory Intel Xeon, které jsou vybaveny čipy FPGA [31].

Tento framework sestává z ovladačů, knihoven a příkazů pro práci s FPGA. OPAE ovladače mapují zdroje FPGA do abstrakce operačního systému. Dále se starají o zpřístupnění těchto zdrojů pro userspace aplikace. Další důležitou prací ovladačů je správa napětí, teploty nebo chyb v hardware [31].

Kromě ovladačů existuje i uživatelské rozhraní napsané v jazyce C, které slouží pro tvorbu aplikací, které interagují s FPGA. Narozdíl od ovladačů, ke kterým se přistupuje v každém operačním systému jinak, se toto API v tomto ohledu nijak nemění [31].

Poslední důležitou částí OPEA frameworku jsou příkazy pro práci s FPGA. Prvním z nich je `fpgasupdate`, který slouží pro nahrávnání nového firmware. Příkaz `fpgainfo` vyčítá informace o fpga, které shromažďují ovladače. Pomocí něj se dá zjistit například

teplota desky, hodnoty napětí na desce, nebo hardwarové chyby. Příkaz `fpgastats` slouží pro vyčítání dat ze statistických registrů MAC vrstvy, jako je například počet přijatých nebo odeslaných rámců. Dalšími OPAE příkazy jsou `fpgadiag` a `fpgabist`, které se používají pro testování FPGA [27].

2.3.2 XL710

XL710 je čip od firmy Intel, který dokáže fungovat jako samostatná síťová karta, nicméně je možné ji použít jako součást jiné karty, jako je tomu u PAC N3000. Pro tento čip existuje ovladač `i40e` v operačním systému Linux i ve frameworku DPDK. Tento ovladač je pak použit i v rámci N3000. Tento čip podporuje standard SR–IOV, díky čemuž je PAC N3000 vhodný kandidát na akceleraci OvS. Každé zařízení XL710 je schopné vytvořit až 128 virtuálních funkcí [28, 27].

Tato karta je schopna poskytnout offload jednoduchých VLAN pravidel pro jakékoliv hardwarové funkce zařízení. Jedná se buď o nastavení VLAN filtru (filtrování rámců s hlavičkou předem specifikovaného VLAN id), nebo přidání resp. odebrání VLAN hlavičky [28].

Jak již zde bylo zmíněno, XL710 disponuje ovladačem `i40e` pro operační systém Linux. Tento ovladač je psaný pro široké použití a podporuje karty jako jsou například X710, V710, X722 a další. Zdrojové kódy tohoto ovladače jsou k dispozici na stránkách společnosti Intel, nicméně samotný ovladač je také součástí Linuxového jádra [25].

Ovladač `i40e` je schopen vytvářet SR–IOV virtuální funkce. Dále je schopen zavádět offload výše zmíněných VLAN pravidel a to pro fyzickou funkci i pro funkce virtuální. Pro tyto virtuální funkce existuje speciální ovladač `iavf`. Jelikož virtuální funkce nemá taková práva na konfiguraci zařízení jako funkce fyzická, byl v tomto systému ovladačů implementován mechanizmus zasílání zpráv. Pokud se například uživatel virtuální funkce rozhodne nastavit přidávání VLAN hlavičky pro všechny rámce, které jsou vysílány z tohoto zařízení, zašle se zpráva z VF do PF. V této zprávě bude žádost o offload tohoto VLAN pravidla a všechny potřebné parametry. Ovladač `i40e` fyzické funkce nejprve zhodnotí, zda provede požadovaný offload (uživatel si může nastavit, že dané požadavky od virtuálních funkcí vyřizovat nechce). Následně provede požadovanou akci, což je v tomto případě VLAN offload. Nakonec ovladač fyzické funkce zašle zprávu o potvrzení vykonané akce a výsledek operace. Ovladač virtuální funkce pak tuto zprávu přijme a považuje tuto akci za vyřízenou. Pokud by žádnou odpověď nedostal, pořád by na tuto zprávu čekal a nefungoval by korektně. Z tohoto důvodu je nutné, aby při běhu aplikace, která používá virtuální funkci, běžel i ovladač fyzické funkce [26, 25].

2.4 Firmware pro akceleraci OvS

V rámci sdružení CESNET a jeho oddělení 707 je vyvíjen FPGA firmware pro akceleraci OvS. Tento firmware je založen na jazyce P4 a funguje na principu `match-action` tabulek. V těch je pevně definováno, jaké položky paketu mohou být porovnávány a jaké akce nad nimy mohou být vykonávány. Za běhu je možné do firmware nahrát pravidla, která už definují konkrétní hodnoty položek a konkrétní akce, které se vykonají v případě shody. Architektura tohoto firmware se skládá z parseru, `match-action engine` a deparseru. V parseru se získají potřebné položky z hlaviček paketů, ty se následně porovnánají v `match-action engine` se vzorovými položkami jednotlivých tabulek. Pokud u některé z tabulek dojde ke shodě, tak se daný paket aplikuje akce, která je definovaná v rámci `action` části tabulky. Pořadí zpracování paketu jednotlivými tabulkami je předem dáno. Po zpracování poslední

tabulkou se paket opět složí v deparseru a pokračuje na patřičný výstup (ethernet, DMA kanál, ...) [36].

Protokolová hlavička	Položky
Ethernet	ethertype zdrojová MAC adresa cílová MAC adresa
VLAN	pcp vid ethertype
IPv4	zdrojová adresa cílová adresa ip_proto ip_dscp ip_ecn ip_ttl
IPv6	zdrojová adresa cílová adresa ip_proto ip_dscp ip_ecn ip_ttl
TCP	zdrojový port cílový port flags
UDP	zdrojový port cílový port

Tabulka 2.1: Položky podporovaných protokolových hlaviček v `table0`

Tabulky v **`match-action engine`** pouze definují, jaká pravidla se do nich mohou vkládat, ty se pak vkládají za běhu. Hlavní **`match-action`** tabulka tohoto firmware se jmenuje `table0` a implementuje všechna podporovaná pravidla, které je možné do tohoto firmware offloadovat. V **`match`** části jsou definované všechny podporované položky paketů, které je možné porovnávat. Tyto položky jsou vyjmenovány v následující tabulce 2.1. V **`match`** části se pak ještě vyskytují dvě položky, které ovšem nejsou součástí standartního paketu. Jedná se o položku `ingress_port`, která slouží pro identifikaci vstupního portu (jeden z ethernetových portů, nebo jeden z DMA kanálů), a položku `selection_id_in`, což je specifický VLAN tag, který se používá k identifikaci zařízení na XL710 v rámci karty PAC N3000. Všechny tyto položky se porovnávají za pomoci bitových masek, které určují, jaké bity v položce se mají porovnávat. Díky tomu je možné porovnávat celou položku, její část, nebo danou položku z porovnávání zcela vyřadit [19].

Část **`action`** `table0` obsahuje 3 základní akce, z nich si uživatel při nahrávání pravidla volí jednu, ta se pak při shodě (**`match`**) nad paketem vykoná. Tyto akce se jmenují `main_action`, `count_and_drop` a `Drop`. Všechny tyto základní akce se dělí na primitivní akce, které představují konkrétní příkaz, který se nad paketem vykoná [19].

Akce `Drop` sestává z jediné primitivní akce `drop`, která způsobí zahození paketu. Akce `count_and_drop` sestává ze dvou primitivních akcí, které jsou, jak už název napovídá,

drop a count. Primitivní akce count inkrementuje čítač, který je následně možné vyčíst a zjistit, kolik paketů se shoduje se vzorem definovaným v match [19].

Základní akce main_action obsahuje mnoho primitivních akcí pro modifikaci jedné položky z paketu. Položky, které se dají takto editovat, se shodují s těmi, které jsou uvedeny v tabulce 2.1. Jedinou výjimku zde tvoří položka TCP flags, která v rámci akce tabulky main_action editovat nelze. Další dvě specifické položky k editaci jsou egress_port, selection_id_out, které jsou výstupní alternativou k položkám ingress_port a selection_id_in v match části. Pro tyto primitivní akce se při nahrání pravidla specifikuje pomocí parametrů nejen nová hodnota položky, ale také bitová maska, která určuje bity, které se při shodě přepíšou. To znamená, že je možné upravit pouze část dané položky a nebo položku vůbec neupravovat. Takže pomocí main_action je možné upravit třeba jen jednu položku paketu z tabulky 2.1, což je žádoucí, jelikož přepsání všech položek se v praxi moc nepoužívá. Kromě editace položek je možné v main_action položku dekrementovat, podporované položky pro tuto akci jsou IPv4 ttl a IPv6 hop limit. Poslední primitivní akcí main_action je count [19].

Specifické použití u karty PAC N3000

Karta PAC N3000 se od běžných FPGA karet odlišuje svými XL710 zařízeními. Díky nim je spojení s hostitelským strojem vyřešeno a v FPGA čipu není nutné implementovat DMA kanály a zabírat tak zdroje, které mohou být využity například pro table0. Použití XL710 ovšem ztěžuje mapování SR-IOV hardwarových funkcí. Ty totiž nejde v rámci firmware přímo adresovat, jelikož jsou zde rozlišovány pouze jednotlivá XL710 zařízení a rozdělení na hardwarové funkce zůstává skryté.

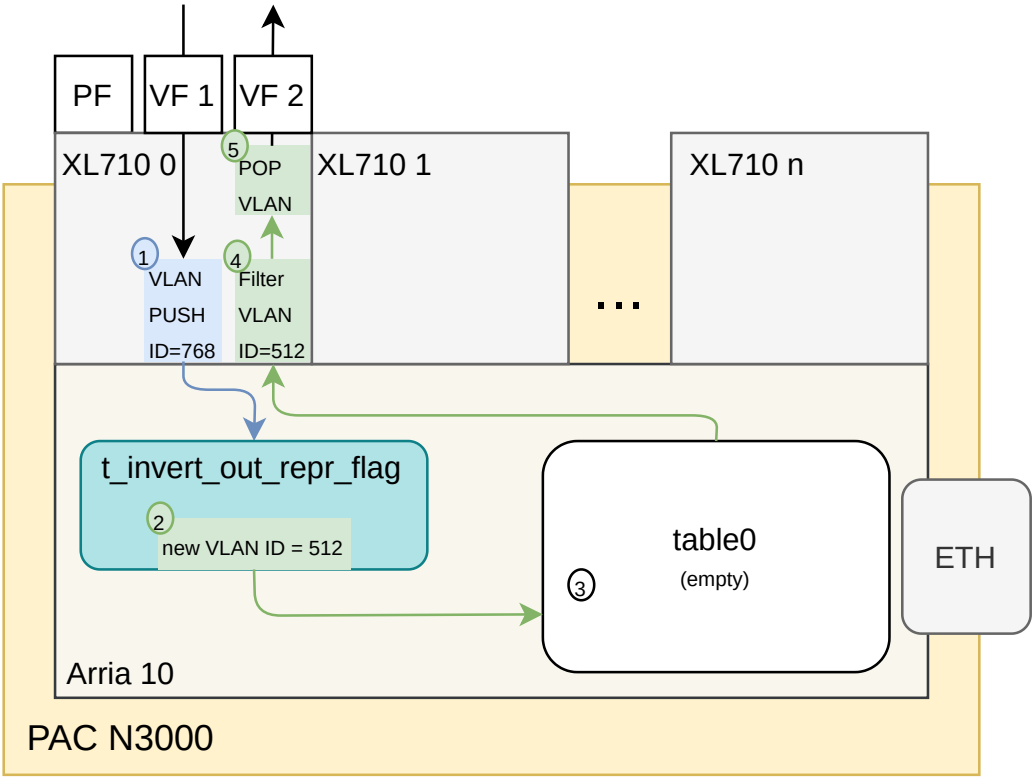
Proto se ve verzi tohoto firmware pro kartu N3000, používá pro identifikaci virtuálních funkcí hlaviček VLAN. Firmware zde implicitně předpokládá, že nejkrajnější VLAN hlavička (*most outer*) v paketu je specifiký tag, který se použije pro identifikaci hardwarové funkce. Tento tag se pak dá použít v match části pod názvem selection_id_in a v action části je k němu možné přistoupit pomocí selection_id_out a pozměnit tak cílovou destinaci paketu.

Před použitím karty PAC N3000 s tímto firmwarem je potřeba nastavit u hardwarových funkcí XL710 přidávání VLAN hlaviček pro odchozí pakety a odebírání VLAN hlaviček pro příchozí pakety. Pro mapování jednotlivých virtuálních funkcí se pak použije VLAN filteru v XL710.

Pro rozlišení paketů, které směřují z FPGA a které směřují do FPGA, existuje ve firmware pomocná tabulka t_invert_out_repr_flag, která invertuje devátý bit (0x100). Pokud by pak byl do FPGA zaslán paket s VLAN id 512 (0x200) vrátil by se z FPGA s VLAN id 768 (0x300).

Na obrázku 2.8 je znázorněna cesta paketu z jedné virtuální funkce do druhé. Před zasláním paketu je v XL710 zařízení nastaveno přidávání VLAN hlavičky s VLAN id 768 pro odchozí pakety z VF 1. Ve VF 2 je nastaven VLAN filter, který propustí pouze pakety s VLAN id 512 a zároveň je nastaveno odebírání této hlavičky, která po filtraci již ztrácí význam. V tabulce table0 není nahrané žádné pravidlo, které by mohlo měnit cestu paketu. V t_invert_out_repr_flag je nahrané výchozí pravidlo, které invertuje devátý bit. Paket v prvním kroku získá VLAN hlavičku v id 768 a následně putuje do FPGA. V druhém kroku se mu změní v t_invert_out_repr_flag VLAN id na 512. Jelikož je tabulka table0 prázdná, paket je nezměněn a pokračuje dál směrem do XL710. Ve

čtvrtém kroku je paket na základě filteru poslán do VF 2, pak je mu v rámci pátého kroku odejmuta hlavička a je směrován do software.



Obrázek 2.8: Cesta paketu z VF 1 do VF 2 pomocí vlan mapování

2.4.1 Knihovna `libp4dev`

Pro vkládání pravidel do firmware pro akceleraci OvS se používá knihovna `libp4dev`, která je také vyvíjena sdružením CESNET. Zde existuje datový typ `p4device_t`, který v software reprezentuje daný firmware. Při práci s firmware je důležité jej nejprve inicializovat pomocí funkce `p4device_init`, pak je vhodné kartu resetovat, k čemuž slouží funkce `p4device_reset`.

Následně je možné vkládat pravidla, ta se definují pomocí datového typu `p4rule_t`, který definuje celé pravidlo. Tento typ pak obsahuje seznamy hodnot typu `p4key_t` a `p4param_t`. Typ `p4key_t` definuje hodnotu položky paketu v `match`, zatímco `p4param_t` slouží pro nastavení parametrů části `action`. Podle těchto parametrů se pak rozhoduje, které primitivní akce se nad paketem v případě shody použijí a jakým způsobem se provedou. Například pro primitivní akci, která nastavuje IPv4 zdrojovou adresu, se pomocí `p4_param_t` definuje, jestli se má tato akce použít a jaká má být nově nastavená zdrojová IPv4 adresa.

Definované pravidlo pomocí typu `p4rule_t` se dá vložit do hardware voláním funkce `p4table_insert_rule`. Takové pravidlo se dá smazat z dané tabulky, k čemuž slouží funkce `p4table_delete_rule`. Knihovna `libp4dev` také umožňuje vyčítat z hardwarových čítačů, které se používají v rámci primitivní akce `count`. Pro reprezentaci ta-

kových čítaču slouží datový typ `p4counter_t` a je možné jej vyčíst voláním funkce `p4counter_read`.

Knihovna `libp4dev` je vytvořena pro síťové karty s čipem FPGA a může fungovat s jakoukoliv kartou, která podporuje tento akcelerační firmware. Pro karty z platformy NDK, které používají `nfb` linux kernel driver funguje bez jakékoliv nutné konfigurace, jelikož podporuje rozhraní `nfb` ovladače. Pokud je potřeba použít `libp4dev` na nějaké neznámé platformě, jen nutné předat knihovně ukazatele na funkce (callbacks) pro interakci s hardware. Tyto ukazatele na funkce se ukládají pomocí typu `p4fw_callback_db_t`, který definuje rozhraní jednotlivých funkcí. Seznam jednotlivých funkcí a jejich účel je znázorněn v tabulce 2.2.

Název funkce	Popis
`init`	Inicializace zařízení
`free`	Dealokace paměti alokované při `init`
`read32`	Přečtení 32-bitového slova z firmware na dané adrese
`write32`	Zapsání 32-bitového slova do firmware na danou adresu
`read32_multi`	Přečtení předem specifikovaného počtu 32-bitových slov z firmware na danou adresu
`write32_multi`	Zapsání předem specifikovaného počtu 32-bitových slov do firmware na danou adresu

Tabulka 2.2: Funkce definované v `p4fw_callback_db_t`

2.5 DPDK

Data Plane Development Kit, zkráceně DPDK, je framework pro vývoj vysokorychlostních síťových aplikací, který se zaměřuje na rychlé zpracování paketů.

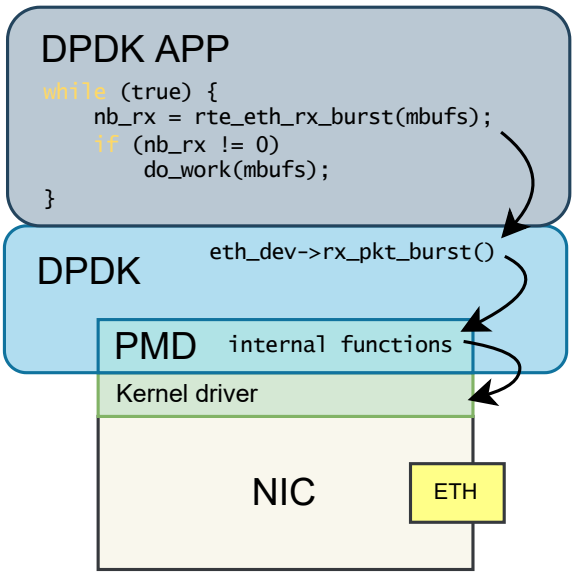
DPDK se používá jako alternativa vůči standardnímu síťovému rozhraní. Tento framework klade důraz na práci v userspace, která minimalizuje počet systémových volání, které zpomalují aplikaci. Dalšími typickými rysy DPDK jsou `polling` a `hugepages`, které jsou použity místo klasických stránek. `Polling` je technika, při níž je zařízení neustále dotazováno, zda nebyly přijaty nové pakety. Tato technika sice spotřebuje 100% CPU, nicméně zamezuje vzniku přerušení, které mají při vysokých síťových rychlostech značnou režii. Proto všechny DPDK ovladače běží v takzvaném `poll mode`. Velké stránky neboli `hugepages` jsou použity, aby zamezily častým výpadkům TLB, které při častém ukládání paketů mohou nastat [20, 6].

Tento framework podporuje širokou škálu CPU, mezi ně patří například klasické procesory řady x86 od výrobců AMD a Intel, dále je podporvána i architektura POWER9 od firmy IBM nebo jádra typu ARM od společnosti Octeon. Co se síťových karet týče, je podporována řada karet od společnosti Intel, jako například již zmíněné N3000, XL710 nebo i X550. Mezi další výrobce podporovaných karet patří například společnost NVidia, NXP nebo CISCO. Podporované operační systémy jsou Linux, FreeBSD a Windows, nicméně pozornost vývojářů a uživatelů je směrovaná na Linux [17].

V DPDK je použita Enviromental Abstraction Layer (EAL), což je vrstva, která zapouzdřuje specifické vlastnosti jednotlivých prostředí a poskytuje jednotné rozhraní ke zdrojům, jako jsou síťové karty nebo CPU. Služby, které EAL poskytuje jsou například spouštění DPDK, správa logických jader, správa paměti, přístup k sběrnici PCI a další. Při spuštění

DPDK aplikace se argumenty programu předávají funkci pro inicializaci EAL. EAL definuje argumenty, které takto může uživatel definovat až při spuštění aplikace. Mezi takové parametry patří například seznam logických jader, na kterých DPDK aplikace poběží, nebo specifikace síťových karet, které bude DPDK používat [10].

Hlavní knihovny DPDK, které poskytují základní složky potřebné pro tvorbu vysokorychlostních síťových aplikací, se nazývají `core components`. Mezi tyto knihovny patří `librte_mbuf`, které poskytuje datovou strukturu `mbuf` pro ukládání paketů. Další knihovny jsou například `librte_ring`, která poskytuje kruhový seznam, do kterého se mohou ukládat pakety ve struktuře `mbuf` nebo `librte_mempool`, která se stará o správu paměti [15].



Obrázek 2.9: Volání funkcí pro přijímání paketů v DPDK

V DPDK se rozlišují různé typy zařízení. Kromě síťových karet, které jsou reprezentovány strukturou `rte_eth_dev`, se zde vyskytují i zařízení pro kompresi, nebo kryptografii. Zařízení se zde dá definovat i velmi obecně pod strukturou `rte_rawdev`, která se dá využít například pro FPGA. Struktura `rte_eth_dev` je použita pouze v knihovnách a ovladačích DPDK, uživatel k ní přistupuje pomocí indexu, který se nazývá `port_id`. Zařízení, která jsou reprezentována pomocí `rte_eth_dev` struktury, se také nazývají DPDK porty [6, 16, 14].

DPDK ovladadače se nazývají Poll Mode Drivery (PMD) a jak už název napovídá, fungují v `polling` režimu. Poskytují funkce pro příjem a odesílání paketů, pro inicializaci zařízení, pro konfiguraci a jiné. Tyto funkce jsou pak použity v knihovnách DPDK, které uživateli poskytnou stejné rozhraní pro práci s jakoukoliv kartou. Zjednodušené volání funkcí z aplikace až do PMD je naznačeno na obrázku 2.9, také je zde ukázáno, jakým způsobem fungují aplikace používající Poll Mode Drivery [14].

PMD jsou závislé na kernel ovladačích, jelikož potřebují rozhraní pro práci se síťovou kartou. Pro tyto účely je možné použít klasické kernel ovladače daných zařízení a nebo využít obecných ovladačů, které slouží pro přenos zařízení do userspace, jako jsou VFIO, nebo méně robustní UIO. Volbu kernel ovladače specifikuje výrobce dané karty, který může podporovat i více možností, jak svou kartu zapojit do DPDK [13].

Další schopnost Poll Mode Driverů je hardwarový offload. DPDK nabízí několik způsobů přesunu pravidel do síťové karty. První možností je nastavení pravidla při konfiguraci zařízení. Dá se tak například offloadovat RSS (Receive Side Scaling). Další možností je použití RTE flow, což je DPDK knihovna, která je blíže popsána v podkapitole 2.5.1. Poslední možností je použití specifických DPDK funkcí pro offload. Dá se tak nastavit například VLAN filter, popřípadě přidávání/odebírání VLAN hlavičky [14].

Někteří tvůrci PMD, jako jsou například vývojáři ze společností Intel nebo NVidia, přidávají možnost vytvářet reprezentátory SR-IOV virtuálních funkcí. Tyto reprezentátory jsou přes kartu namapovány k virtuálním funkcím a typicky je používá virtuální přepínač, který je má připojené ke svým portům. Reprezentátory se dají použít pro konfiguraci virtuálních funkcí přes DPDK nebo pro síťovou komunikaci s virtuální funkcí. I když jsou reprezentátory vytvářeny hlavním zařízením (`rte_eth_dev`) dané karty, z pohledu DPDK se chovají jako jakékoliv jiné porty [14].

EAL nabízí možnost specifikovat u dané karty i seznam virtuálních funkcí, pro které se mají vytvořit reprezentátory. Pro novou verzi VFIO, která je schopna vytvářet virtuální funkce, má EAL také speciální parametry. Je zde možné specifikovat VFIO VF token a takzvaný `file-prefix`, který určuje, zda bude aplikace fungovat s fyzickou nebo virtuální funkcí. Používání jak fyzické, tak virtuální funkce v jedné DPDK aplikaci není možné [13].

Pro účely testování existuje v DPDK nástroj zvaný **`test-pmd`**, který nabízí spoustu příkazů, kterými je možné testovat funkce dané karty. Tyto příkazy je možné spustit i v interaktivním režimu, který poskytuje příjemné rozhraní pro testování. Mezi tyto příkazy patří například konfigurace karty (VLAN offload, nastavení promiskuitního režimu, nastavení linky, ...), nebo nahrávání RTE flow pravidel. Tento nástroj je také schopen zpracování paketů (**`packet forwarding`**), pro které má k dispozici několik módů. Některé z těchto módů jsou vypsány v tabulce 2.3. Pomocí nastavení hodnoty **`verbosity`** je možné během přeposílání zjišťovat informace o daných paketech, jako jsou například vstupní/výstupní porty nebo MAC adresy [18].

Název módu	Popis
`io`	Přijaté pakety jsou odeslány bez jakékoliv změny
`macswap`	Přijaté pakety jsou odeslány, ještě před tím je ale každému paketu prohozena zdrojová a cílová MAC adresa
`rxonly`	Pakety jsou pouze přijímány, bez odesílání
`txonly`	Probíhá generování paketů, které jsou potom odesílány, žádné pakety nejsou přijímány

Tabulka 2.3: **`packet forwarding`** módy nástroje **`test-pmd`**

2.5.1 RTE flow

RTE flow je DPDK knihovna, která slouží pro přenos klasifikačních pravidel do hardware. Tato knihovna definuje generické API, pomocí kterého je možné nahrávat pravidla do hardware, vyčítat z nich data nebo je mazat [9].

Pravidla se definují pomocí atributů (**`attributes`**), vzoru (**`matching pattern`**) a akcí (**`actions`**). Vzor definuje, jak mají vypadat pakety, nad kterými se budou provádět akce. Atributy poskytují doplňující informace o RTE flow pravidlu [9].

Vzor se skládá z položek (**`items`**), které reprezentují protokolovou hlavičku, nebo nějakým jiným způsobem specifikují klasifikovaný paket (**`meta items`**). Položky vzoru se definují

pomocí políček `spec`, `last` a `mask`. Hodnota položky se specifikuje pomocí `spec`. Pro specifikaci rozsahu vhodných hodnot se kombinuje `spec` s `last`. Například pro zdrojovou IPv4 adresu by nastavení `spec` na 10.0.0.2 a `last` na 10.0.1.0 znamenalo, že ke shodě (`match`) by došlon pokud by zdrojová IPv4 adresa paketu byla v rozsahu od 10.0.0.2 po 10.0.1.0. Políčko `mask` specifikuje, které bity z položky se mají porovnávat. Pro cílovou MAC adresu, kde `spec` je 00:11:17:A3:34:45 a `mask` je FF:FF:FF:00:00:00, by se pak porovnávaly jen první 3 oktety (`vendor id`) [9].

RTE flow definuje přes 40 různých položek, proto jsou zde zmíněny jen ty, které jsou důležité z hlediska této práce. Seznam takových položek je uveden v tabulce 2.4.

Název položky	Popis
`ETH`	Ethernetová hlavička, obsahuje zdrojovou a cílovou MAC adresu, Ether type a `has_vlan` flag, který specifikuje, zda je v paketu přítomná VLAN hlavička
`VLAN`	VLAN hlavička, obsahuje tci, Ether type a `has_more_vlan` flag, který specifikuje, že paket obsahuje ještě alespoň jednu VLAN hlavičku
`IPV4`	IPV4 hlavička, části stejné jako v RFC 760
`UDP`	UDP hlavička, části stejné jako v RFC 768
`TCP`	TCP hlavička, části stejné jako v RFC 793
`PORT_ID`	Označení portu, kterým paket vstoupil do karty (Virtuální funkce, reprezentátor, Ethernetový port, apod.)

Tabulka 2.4: RTE flow položky použité v této práci

RTE flow akce slouží pro úpravu paketu nebo jinou manipulaci s paketem. Jelikož tato knihovna definuje více než šedesát takových akcí, jsou zde zmíněny jen akce důležité z hlediska této práce. Ty jsou vypsány v tabulce 2.5 [9].

Název akce	Popis
`PORT_ID`	Přesměrování paketu do specifikovaného DPDK portu.
`COUNT`	Registrace hardwarového čítače, který se v případě shody (`match`) inkrementuje a za běhu aplikace se dá hodnotu tohoto čítače získat pomocí funkce `rte_flow_query`
`DROP`	Zahození paketu
`DEC_TTL`	Dekrementace TTL (Time To Live)
`SET_<FIELD>`	Nastavení položky paketu `<FIELD>`. Ta může být například zdrojová/cílová MAC adresa, TTL, zdrojová/cílová IPv4 adresa, zdrojová/cílová IPv6 adresa a zdrojový/cílový port transportní vrstvy

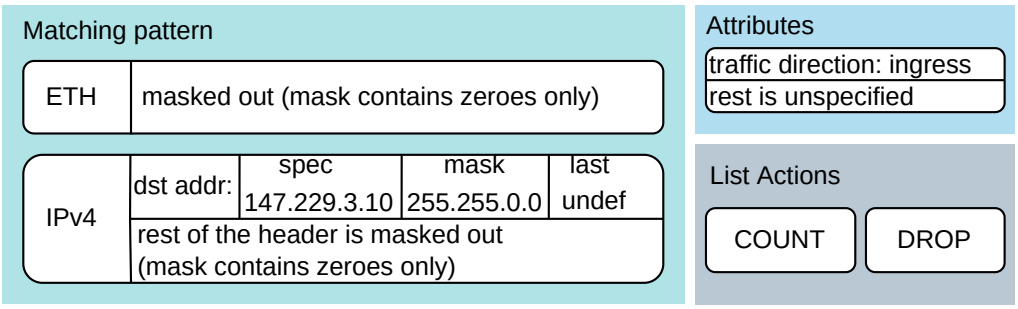
Tabulka 2.5: RTE flow akce použité v této práci

Ohledně akce `PORT_ID` je ještě vhodné zmínit, že v nové verzi DPDK 21.11 byla označena za zastaralou (depraceted) a doporučuje se místo ní používat nově přidané akce `PORT_REPRESENTOR`, `REPRESENTED_PORT`. Jelikož se tato práce soustředí na akceleraci OvS, kde se zatím používá starších verzí DPDK s akcí `PORT_ID`, je zde popsáno rozhraní RTE flow pro starší verze, kde se tato akce vyskytuje [9].

Atributy jsou doplňující informace, pomocí kterých je možné upřesnit DPDK ovladači informace o pravidle. Nejdůležitějším atributem je `traffic direction`, který určuje, zda se pravidla budou na pakety aplikovat před vstupem do klasifikační pipeline (`ingress`), po vstupu z klasifikační pipeline (`egress`) nebo v obou případech. Poslední volba se nedoporučuje a většinou není ani ze strany výrobců síťových karet podporována, ale v některých okrajových případech může být užitečná [9].

Dalším důležitým atributem pro tuto práci je `transfer`. Tento atribut je schopen pozměnit význam některých pravidel. V kombinaci s akcí `PORT_ID` se změní cílová destinace paketu. Pokud byl v `PORT_ID` jako nová destinace definován reprezentátor v kombinaci s atributem `transfer`, paket se zašle do virtuální funkce. Pokud je novou destinací hlavní zařízení, paket se přenese do ethernetového portu [9].

Pro lepší představu o RTE flow pravidlech je na obrázku 2.10 uveden příklad RTE flow pravidla. Toto pravidlo způsobí zahození všech paketů, jejichž první dva oktety cílové IPv4 adresy jsou 147.229. Zároveň se počet všech takto zahozených paketů bude ukládat v čítači, který je možné za běhu vyčíst [9].



Obrázek 2.10: Příklad RTE flow pravidla

Datové struktury, které reprezentují vzor, akce a atributy, se používají jako parametry funkce `rte_flow_create`, která je předá ovladači. Ten na základě tohoto vstupu zhodnotí, zda je možné takové pravidlo přenést do hardware. Pokud to možné je, ovladač toto pravidlo offloaduje. Pro zhodnocení, zda je dané pravidlo možné přenést do hardware, existuje funkce `rte_flow_validate`. Tato akce může být užitečná, protože pomocí RTE flow je možné definovat velmi velké množství pravidel, a sami tvůrci RTE flow přiznávají, že neexistuje síťová karta, která by podporovala všechny z nich. Funkce `rte_flow_query` je schopna vyčítat z karty data o jednotlivých akcích, které takové vyčítání podporují. Pro smazaní pravidla z karty existuje funkce `rte_flow_destroy`. Pokud chce uživatel smazat všechna RTE flow pravidla, která na dané kartě offloadoval, může použít funkci `rte_flow_flush`. Poslední funkcí z rozhraní RTE flow je `rte_flow_isolate`, která slouží pro nastavení izolovaného módu [9].

Všechny tyto funkce RTE flow rozhraní pak volají callbacky ze struktury `rte_flow_ops` patřičného PMD, kde existuje pro každou výše zmíněnou funkci jeden callback. Tyto callbacky jsou pojmenovány podle funkcí, v rámci kterých se volají, například ve funkci `rte_flow_create` se volá callback `create`, ve funkci `rte_flow_destroy` se volá callback `destroy` a podobně [9].

Pro reprezentaci RTE flow pravidla existuje v DPDK struktura `rte_flow`. Ukazatel na tuto strukturu je návratovým typem funkce `rte_flow_create`, která tuto strukturu v rámci svého běhu alokuje. Tento ukazatel se pak používá jako parametr u funkcí `rte_flow_query`, kde se z pravidla vyčítají data, a `rte_flow_destroy`, kde se pravi-

dlo maže a struktura `rte_flow` je dealokována. Zajímavostí na této struktuře je fakt, že je v rámci DPDK knihoven pouze deklarována a její definice závisí už na konkrétním PMD. To znamená, že například v ovladačích `mlx5` (NVidia) a `ixgbe` (Intel) vypadá struktura `rte_flow` odlišně [9].

2.5.2 PMD `i40e`

Poll Mode Driver `i40e` je ovladač pro síťové karty X710, XL710, XXV710 a X722. Tento PMD je možné použít s `vfio` nebo `i40e` linux kernel ovladačem. Schopnosti tohoto ovladače zahrnují VLAN offload, nastavování promiskuitního režimu nebo nastavení ethernetové linky [11].

Tento PMD je schopen vytvářet reprezentátory SR–IOV virtuálních funkcí, ty ovšem nemají `data path`. To znamená, že pomocí `i40e` reprezentátorů není možné posílat a přijímat pakety, což znemožňuje použití tohoto ovladače pro klasickou SR–IOV architekturu akcelerace OvS, která je znázorněna na obrázku 2.3. Zde je nutné, aby mohly být přeposílány pakety mezi virtuálními funkcemi a jejich reprezentátory. Reprezentátory vytvářené `i40e` PMD se prozatím (DPDK verze 21.02) dají použít pouze k částečné konfiguraci virtuální funkce (například nastavení promiskuitního režimu) [11, 7].

Podobně jako jeho linux kernel protějšek, i tento ovladač je schopen komunikovat s virtuálními funkcemi pomocí rozhraní zasílání zpráv. Toto rozhraní je jednotné, a proto je možné komunikovat i například ve scénáři DPDK PF a kernel VF nebo naopak. Tento mechanizmus funguje způsobem žádost (VF) – odpověď (PF). Například žádost o VLAN offload z VF je odpovězeno potvrzením o úspěšném offloadu popřípadě zprávou o chybě nebo odmítnutí [11, 7].

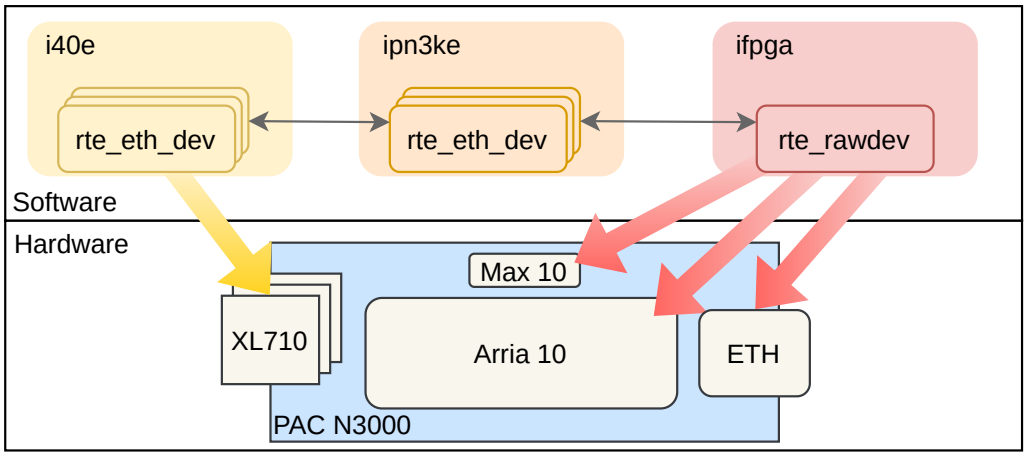
Jak již zde bylo zmíněno, tento ovladač je schopen VLAN offloadu. Zajímavostí tohoto PMD je i možnost nastavovat VLAN offload i pro virtuální funkce. Tato funkcionalita zatím v DPDK nemá standardní rozhraní, a proto jsou tyto funkce definovány specificky pro tento ovladač v souboru `drivers/net/i40e/rte_pmd_i40e.h`. Podobné rozhraní je definováno i pro jiné PMD karet od společnosti Intel a lze předpokládat, že v budoucnu se stanou součástí standardního rozhraní DPDK [11, 7].

Ve složce `drivers/net/i40e` DPDK repozitáře je kromě `i40e` implementován ještě `i40evf` Poll Mode Driver. Tento ovladač se používá pro SR-IOV virtuální funkce a je, jako jeho linux kernel protějšek, založen na PF-VF zasílání zpráv. Tento ovladač obsahuje, narozdíl od `i40e` reprezentátorů, funkce pro zasílání a přijímání paketů. Dále je schopen nastavovat VLAN offload, i když tuto činnost je možné přenechat PF ovladači [7].

2.5.3 PMD `ipn3ke`

PMD `ipn3ke` je DPDK ovladač karty PAC N3000 od společnosti Intel. Pro ovládání této karty však nestačí pouze tento ovladač. Pro XL710 zařízení této karty je použit již zmíněný PMD `i40e`. Pro poskytnutí základních funkcionalit pro práci s FPGA, který v normálních případech zajišťuje OPAE framework, je v DPDK vytvořen `rawdev` ovladač `ifpga`. Ten z OPAE přejímá značnou část zdrojového kódu [12, 7].

Ovladač `ifpga`, je použit jak pro hlavní FPGA Arria 10, tak pro vedlejší Max 10 FPGA karty PAC N3000. Tento ovladač používá Max 10 pro monitování desky a je schopen v případě kritických hodnot přerušit práci s kartou, aby nedošlo k jejímu poškození. Toto přerušení může nastat například při přehřátí karty. Dále tento ovladač poskytuje rozhraní pro práci s FPGA, které zahrnuje inicializaci, resetování nebo ukončení práce s FPGA. Kromě těchto základních funkcí nabízí `ifpga` i možnost nahrát do FPGA nový firmware. Rozhraní



Obrázek 2.11: PMD architektura, která ovládá PAC N3000 v DPDK

tohoto ovladače je používáno v rámci `ipn3ke` pro komunikaci s FPGA firmware. Pomocí něj je implementováno rozhraní RTE flow, které je závislé na specifickém designu od firmy Intel [12, 7].

PMD `ipn3ke` sám o sobě nevytváří plnohodnotná zařízení schopné síťové komunikace. Odesílání a přijímání paketů je plně přenecháno ovladači `i40e`. Ke každému `i40e` zařízení ale vytvoří reprezentátor, který je schopen částečné konfigurace a RTE flow offloadu. Tento reprezentátor nijak nesouvisí s konceptem SR–IOV, kde se používají pro napojení na virtuální funkce. Místo toho je propojen s jedním `i40e` zařízením (tj. fyzická funkce XL710) [12, 7].

Na obrázku 2.11 je vidět celá architektura ovladačů, která je potřeba pro komunikaci s kartou. Je zde znázorněna i komunikace mezi nimi. Pokud je potřeba přístup ke zdrojům FPGA nebo například k MAC komponentám, komunikaci poskytuje ovladač `ifpga`. V případě XL710 vše zastřešuje `i40e`. Ovladač `ipn3ke` slouží jako prostředník mezi ovladači navzájem a mezi `ifpga` a uživatelem, který chce používat FPGA například pro RTE flow [12, 7].

Vzhledem k složitější architektuře této karty je i její spouštění v DPDK složitější. Nejprve je nutné na celou kartu (čipy XL710 a FPGA) připojit ovladač VFIO a následně se musí jako celek předat DPDK aplikaci. Jediné části karty N3000, které mohou být předány odděleně, jsou virtuální funkce, které vznikají v rámci konceptu SR–IOV [12, 7].

Kapitola 3

Praktická část

3.1 Návrh a Implementace

V rámci praktické části je nutné navrhnout, implementovat a otestovat akcelerační prototyp. Základem tohoto prototypu je karta N3000, do které je nahrán akcelerační firmware od společnosti CESNET. Virtuální stroje jsou napojené na virtuální funkce této karty (respektive čipu XL710, který je její součástí) a zbytek karty je předán DPDK. OvS pak používá tuto část skrze DPDK a zároveň využívá rozhraní RTE flow pro svou akceleraci. Toto rozhraní je implementováno na základě akceleračního firmware.

Implementaci a návrh tohoto prototypu je možné rozdělit na čtyři kroky. Nejprve je nutné zajistit podporu `libp4dev` v DPDK ovladači karty N3000 3.1.1. Po dokončení této části je možné používat v DPDK kartu N3000 s akceleračním firmware a zároveň existuje rozhraní pro nahrávání pravidel do `match-action engine`. Dalším krokem je zajištění funkčního SR–IOV síťového rozhraní, po kterém jsou pakety z virtuálních strojů zaslány do příslušných OvS portů a naopak 3.1.2. Třetím krokem je implementace RTE flow rozhraní, pomocí kterého je možné přesunout klasifikační pravidla OvS na úroveň karty N3000 3.1.3. Díky tomuto kroku se daná pravidla budou vykonávat v hardware, což sníží zátěž CPU a zvýší propustnost OvS. Posledním krokem je vytvoření prostředí pro práci s prototypem, které zajistí jeho pohodlnější spouštění 3.1.4.

3.1.1 Portace knihovny `libp4dev` do PMD `ipn3ke`

Pro práci s akceleračním firmware se používá knihovny `libp4dev`. Jelikož karta PAC N3000 pod ovladačem VFIO nepatří mezi podporované platformy `libp4dev`, je potřeba nejprve implementovat funkce pro práci s touto platformou. Následně se tyto funkce mohou předat pomocí `p4fw_callback_db_t` knihovně `libp4dev` při inicializaci. Pro tvorbu těchto funkcí je možné využít již implementovaných funkcí v PMD `ipn3ke`, které využívá ovladač `ifpga` pro komunikaci s FPGA.

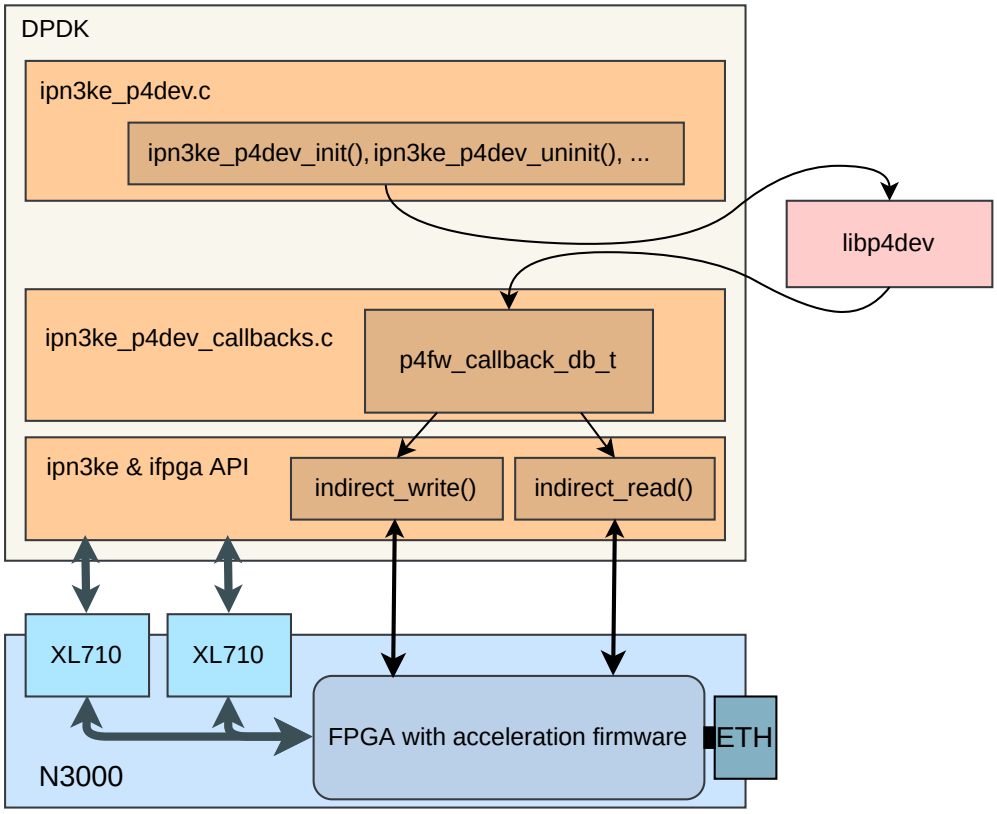
Po zajištění podpory této platformy v `libp4dev` je možné rozšířit ovladač `ipn3ke` tak, aby podporoval práci s tímto firmware. Při inicializaci toho ovladače je možné přečíst UUID firmware v FPGA, podle toho zle zjistit, zda je nahraný firmware od společnosti CESNET. V takovém případě je možné inicializovat tento firmware pomocí `libp4dev` a tím zahájit práci s kartou.

Implementace

Pro implementaci byla zvolena DPDK verze 21.02, jelikož zde vývojáři již stihli zareagovat na novou verzi VFIO, ve které je možné vytvářet VFIO virtuální funkce. Druhým důvodem pro výběr této verze je podpora ze strany OvS (novější verze DPDK než je 21.02 v době návrhu a implementace ještě nebyly do OvS zaintegrovány).

První změnou bylo zprovoznění `libp4dev` v ovladači `ipn3ke`. Pro tyto účely byly vytvořeny funkce zajišťující přístup k FPGA, které byly použity pro vyplnění struktury `p4fw_callback_db_t`. Tyto funkce jsou nepřímo závislé na rozhraní poskytované ovladačem `ifpga`, jelikož je pro zápis a čtení dat z paměťového prostoru karty použita adresa získaná z tohoto ovladače. Pro implementaci těchto funkcí byl vytvořen nový soubor `ipn3ke_p4dev_callbacks.c` a zároveň byl vytvořen i další nový soubor `ipn3ke_p4dev.c`, který je určen pro interakci s `libp4dev` a akceleračním firmwarem.

Pro tuto interakci byla implementována struktura `ipn3ke_p4dev`, která reprezentuje akcelerační firmware v rámci ovladače. Ta obsahuje `libp4dev` identifikátor `p4device_t`, zámek typu `spinlock` a pomocnou strukturu `counter_info`. Zámek zde slouží pro vyloučení vzájemného přístupu více jader, které je potřeba uplatnit při práci s knihovnou `libp4dev`. Struktura `counter_info` obsahuje `p4counter_t`, který zajišťuje přístup k hardwarovým čítačům, které se používají v kombinaci s primitivní akcí `count`. Dále tato struktura slouží k rezervaci čítačů, které je nutné v rámci každého pravidla s akcí `count` registrovat a posléze také při mazání pravidla odregistrovat. Pro tyto účely tato struktura obsahuje bitové pole o velikosti počtu čítačů, kde každý bit značí čítač, který je buď zaregistrovaný (stav 1) nebo volný (stav 0).



Obrázek 3.1: Závislosti při přístupu do firmware

Nejdůležitější funkcí pro práci s akceleračním firmwarem v souboru `ipn3ke_p4dev.c` je `ipn3ke_p4dev_init`. Ta se stará o inicializaci `libp4dev`, `match-action engine` a struktury `ipn3ke_p4dev`. Jako první je zde alokována struktura `ipn3ke_p4dev`. Následně je inicializována knihovna `libp4dev` a jsou jí předány funkce pro přístup k FPGA ve struktuře `p4fw_callback_db_t`. Poté je inicializován a resetován `match-action engine`. Během těchto rutin je nastaven identifikátor `p4device_t` ve struktuře `ipn3ke_p4dev`. Dalším krokem je nahrání výchozích pravidel do `match-action` tabulek, která určují, co se s paketem stane, pokud nebude vyhovovat žádnému pravidlu. Posledním krokem inicializace je povolení chodu `match-action engine`, což znamená, že od tohoto momentu je karta schopna přijímat a vysílat pakety. Dále se tato funkce stará o inicializaci alokované struktury `ipn3ke_p4dev`. To zahrnuje získání identifikátoru pro práci s čítači `p4counter_t`, vynulování bitového pole pro rezervaci čítačů a inicializaci zámku, který bude použit při přístupu k `libp4dev`.

Inicializační rutiny `ipn3ke` ovladače byly upraveny tak, aby se v případě zjištění UUID akceleračního firmware zavolala funkce `ipn3ke_p4dev_init`. Po skončení této funkce se struktura `ipn3ke_p4dev` propaguje do `ipn3ke` reprezentátorů, aby mohly za běhu aplikace využívat rozhraní `libp4dev`, což je potřebné pro poskytnutí RTE flow (3.1.3).

Pro ukončení práce s `libp4dev` byla implementována funkce `ipn3ke_p4dev_uninit`. V té se řeší zejména uvolňování paměti, která byla alokována pro `ipn3ke_p4dev` a další `libp4dev` struktury. Deinicializační rutiny ovladače `ipn3ke` byly upraveny tak, aby se při nich volala tato funkce.

Na obrázku 3.1 jsou naznačeny závislosti při přístupu do paměťového prostoru firmware. Ve funkcích, jako je například `ipn3ke_p4dev_init`, se volají funkce `libp4dev`, ty používají rozhraní nastavené v `p4fw_callback_db_t`. V tomto rozhraní se používají funkce pro nepřímý přístup do hardware, které jsou součástí ovladače `ipn3ke`, a zároveň se zde používá adresa získaná z ovladače `ifpga`.

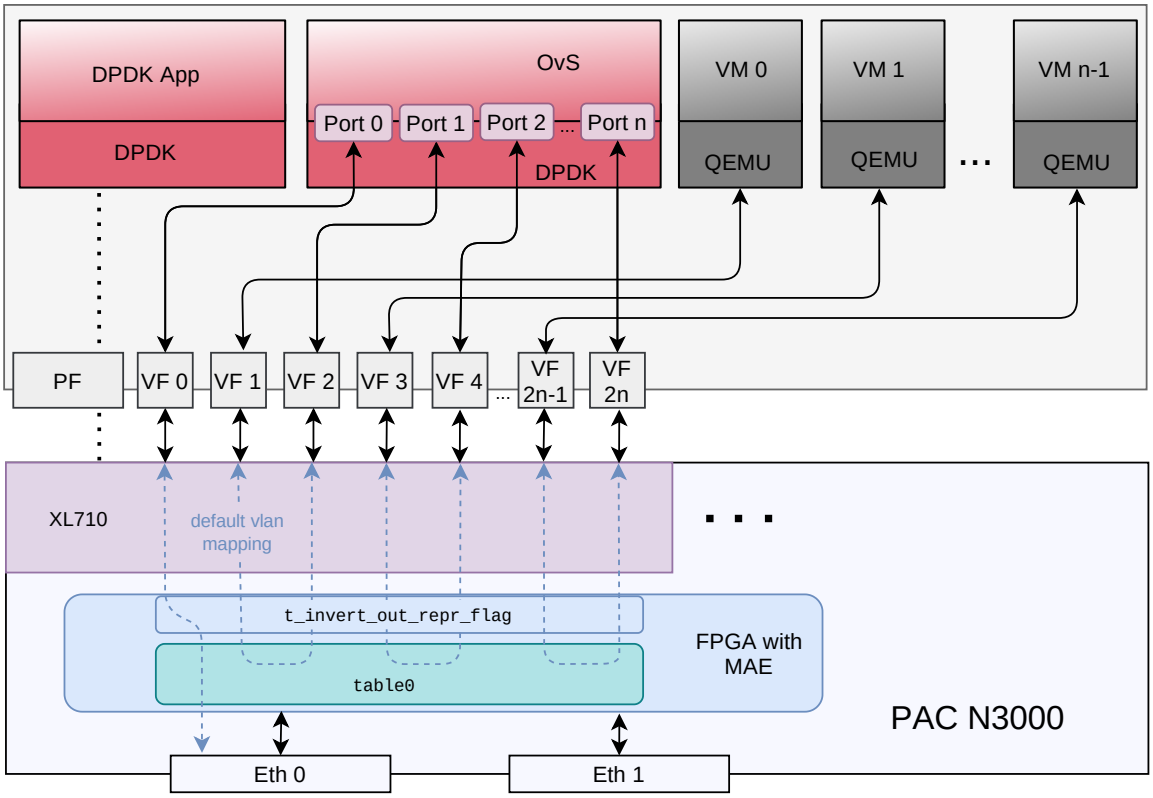
3.1.2 Zajištění funkčního SR–IOV síťového rozhraní

Dalším důležitým krokem je zajištění tvorby SR–IOV virtuálních funkcí. Tyto funkce je možné vytvářet v rámci linux kernel ovladačů `i40e` nebo VFIO. Aby karta fungovala v prostředí DPDK, je nutné ji celou (jak FPGA tak XL710) předat ovladači VFIO. Ten podporuje tvorbu SR-IOV virtuálních funkcí až v novějších verzích linuxového jádra (5.7 a výše).

Dále je nutné se zabývat volbou vhodného hypervizoru, který je schopen zajistit tvorbu virtuálních strojů s možností připojit k nim VFIO virtuální funkce. Z tohoto důvodu byl zvolen hypervizor QEMU. Jedná se o velmi častou a stabilní volbu v linuxovém prostředí, která v kombinaci s KVM nabízí dobrý výkon. Vývojáři QEMU v době návrhu a implementace této práce nevydali verzi s podporou VFIO virtuálních funkcí. Nicméně již existuje patch soubor, pomocí kterého je možné v QEMU předat VF token do VFIO, a tak zajistit podporu virtuálních funkcí tohoto ovladače.

Nyní je vhodné zamyslet se nad zapojením karty N3000 do OvS–DPDK. Jelikož `i40e` reprezentátory nemají `data path` (viz podkapitola 2.5.2), byly pro tyto účely zvoleny samotné virtuální funkce. To znamená, že polovina funkcí bude použita pro virtuální stroje a druhá polovina bude připojena na porty OvS–DPDK. Jelikož v DPDK (verze 21.02) není možné v rámci jednoho programu používat VFIO fyzické a virtuální funkce zároveň, je nutné pro OvS–DPDK použít jen virtuální funkce. Fyzické funkce je nutné spustit v odděleném programu, jelikož VFIO virtuální funkce není možné využívat, pokud neběží aplikace s fyzickou funkcí (viz podkapitola 2.2.2).

K využití karty pro práci s OvS je nutné namapovat virtuální funkce na své reprezentátory (v tomto případě další virtuální funkce) tak, aby bylo možné přeposílat pakety virtuální funkce do jejího reprezentátoru a naopak. Pro toto mapování je možné použít VLAN hlaviček, které jsou přidávány, odebírány a filtrovány v zařízeních XL710 tak, jak je popsáno v kapitole 2.4. Pro vkládání VLAN pravidel do XL710 je možné použít rozhraní PMD `i40e` v souboru `drivers/net/i40e/rte_pmd_i40e.h`.



Obrázek 3.2: Navržená architektura pro akceleraci OvS

Tato navržená architektura je znázorněna na obrázku 3.2. Pro jednoduchost je zde nakresleno jen jedno XL710 zařízení, nicméně všechny další XL710 budou fungovat stejným způsobem.

Kromě způsobu samotného mapování je také nutné se zabývat načasováním. Výše zmíněná architektura totiž zahrnuje ovladače XL710 virtuálních funkcí, které fungují nezávisle na sobě. PMD `i40evf` při inicializaci přemazává část konfigurace XL710 a podobně tak činí i linux kernel ovladač `iavf`, který se používá ve virtuálních strojích. Z tohoto důvodu je nutné, aby VLAN mapování bylo provedeno až po spuštění, inicializaci a konfiguraci OvS a všech virtuálních strojů.

Implementace

Pro mapovaní SR–IOV funkcí byla vytvořena DPDK aplikace s názvem `vlan-mapper` a byla umístěna v DPDK repozitáři v nové složce `app/vlan-mapper`. Kostra programu a soubory sloužící pro překlad (`Makefile` a `meson.build`) byly přejaty z DPDK aplikace, která je uložena ve složce `examples/skeleton`. Aplikace `vlan-mapper` předpokládá, že jí bude pomocí EAL argumentů přidělena alespoň jedna `i40e` fyzická funkce, pomocí které je

možné provést VLAN mapování. Pokud by této aplikaci byla přiřazena jiná zařízení, byla by při mapování ignorována.

Tato aplikace po spuštění inicializuje přidělená DPDK zařízení a následně se přepne do interaktivního režimu. Zde je možné zadávat příkazy podobně jako například v DPDK aplikaci `test-pmd`, tyto příkazy jsou uvedeny v tabulce 3.1. Tento interaktivní režim byl vytvořen zejména proto, aby bylo umožněno uživateli této aplikace rozhodovat o načasování jednotlivých příkazů. To je velmi důležité u příkazu `map_vlan`, jelikož při nesprávné době mapování (například při inicializaci) by byla část mapování přemazána.

Název Příkazu	Popis
`map_vlan`	Zahájení VLAN mapování.
`get_vfs_stats`	Získání statistik o DPDK zařízeních.
`help`	Zobrazení nápovědy.
`end`	Ukončení aplikace.

Tabulka 3.1: Příkazy aplikace `vlan-mapper`

Pro VLAN mapování byla implementována funkce `map_i40e_vfs`, ta se volá v rámci příkazu `map_vlan` pro všechny dostupné `i40e` fyzické funkce. Zde se používá rozhraní ovladače `i40e` ze souboru `rte_pmd_i40e.h`, které zahrnuje funkce pro VLAN vkládání, odebírání a filtrování.

Pro úspěšné mapování je také nutné znát počet virtuálních funkcí v rámci daného XL710 zařízení. PMD `i40e` tuto informaci samozřejmě obsahuje, nicméně prozatím neexistuje rozhraní, pomocí kterého se dá tento počet zjistit. Z tohoto důvodu byla v tomto ovladači implementována funkce `rte_pmd_i40e_get_vf_number` a zařazena do rozhraní v souboru `rte_pmd_i40e.h`.

Funkce `map_i40e_vfs` nejprve zjistí počet virtuálních funkcí a následně je rozdělí do dvojic (pokud je jich lichý počet, je poslední virtuální funkce mapována na ethernetový port). Tyto dvojice jsou na sebe následně namapovány tak, aby jedna z nich mohla být připojena k virtuálnímu stroji a druhá sloužila jako reprezentátor v OvS. Pro každou dvojici virtuálních funkcí `(VF1, VF2)` jsou použity dva VLAN identifikátory `vid1` a `vid2`, kde každý slouží pro jeden směr komunikace. To znamená, že `vid1` je použit ve směru `VF1 -> VF2` a `vid2` ve směru `VF2 -> VF1`. Tyto identifikátory jsou pak použity pro mapování stejným způsobem, jako je popsáno v kapitole 2.4 a znázorněno na obrázku 2.8. Pro oba identifikátory také platí, že jsou unikátní v rámci celého běhu aplikace `vlan-mapper`.

Příkaz `get_vfs_stats` byl implementován pro možnost získání statistik o virtuálních funkcích, které pomohou při implementaci, ladění a předvádění prototypu. Statistiky obsahují počty přijatých a odeslaných paketů z jednotlivých virtuálních funkcí.

Příkaz `end` způsobí korektní ukončení aplikace. To by mělo nastat po ukončení všech ostatních aplikací, které využívají virtuální funkce, tedy OvS a virtuální stroje. V opačném případě by se virtuální funkce dostaly do stavu, ve kterém se nedají použít.

Během implementace VLAN mapování byl zjištěn bug v linux kernel ovladači pro XL710 virtuální funkce `iavf`. Bug se projevoval tak, že přijaté pakety z tohoto zařízení v software pořád obsahovaly VLAN hlavičku, i když bylo před tím nastaveno její odstraňování. Toto nastavení bylo prováděno mnoha způsoby jak ze strany DPDK pomocí aplikací `vlan-mapper` a `test-pmd`, tak ze strany virtuálního stroje pomocí příkazu `ethtool`. Žádné z těchto nastavení nemělo na odstraňování VLAN hlaviček vliv. Tento problém byl vyřešen instalací nové verze ovladače `iavf` s pozměněným zdrojovým kódem. Tato drobná úprava spočívala v nastavení odebírání hlavičky VLAN již během inicializace.

3.1.3 RTE flow

Posledním krokem je návrh a implementace RTE flow. Toto rozhraní doplňuje již funkční OvS architekturu o možnost přenést klasifikační pravidla na úroveň síťové karty, čímž šetří výkonu CPU a zvyšuje propustnost OvS. Principem tohoto rozhraní je převod pravidel ve formátu RTE flow na `libp4dev` reprezentaci pravidla (`p4rule_t`) a jeho následné nahrání do akceleračního firmware (konkrétně do tabulky `table0`). Tato tabulka byla navržena právě pro akceleraci OvS, a proto je možné do ní nahrát většinu RTE flow pravidel, které OvS vyžaduje.

Podporovaná RTE flow pravidla jsou určena podporovanými atributy, vzorem a akcemi, ze kterých se toto pravidlo musí skládat. Navržené RTE flow rozhraní podporuje atributy `ingress` a `transfer`. Přičemž `ingress` je v pravidle vyžadován, jelikož určuje jediný způsob klasifikace, který je pomocí akceleračního firmware možný. Podporované položky vzoru a akce jsou vypsány v následujících odrážkách.

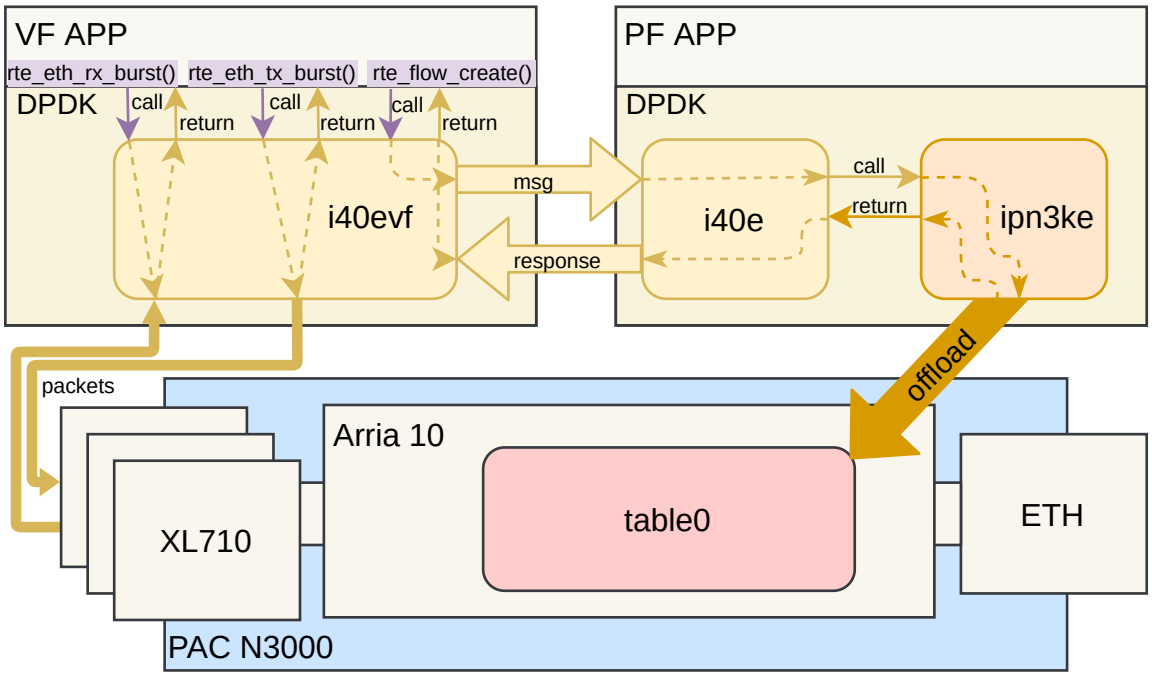
- Podporované položky RTE flow vzoru:
  - `ETH`
  - `VLAN`
  - `IPV4`
  - `UDP`
  - `TCP`
  - `PORT_ID`
- Podporované RTE flow akce:
  - `PORT_ID`
  - `COUNT`
  - `DROP`
  - `DEC_TTL`
  - `SET_MAC_SRC`
  - `SET_MAC_DST`
  - `SET_TTL`
  - `SET_IPV4_DST`
  - `SET_IPV4_SRC`
  - `SET_TP_DST`
  - `SET_TP_SRC`

Poskytnutí RTE flow rozhraní bylo navrženo pro dva DPDK ovladače. Prvním z nich je PMD `ipn3ke`, který se jeví jako nejvíce logická volba. Pro nahrávání RTE flow pravidel do FPGA by měl sloužit právě tento ovladač a většině aplikacím by přišlo použití tohoto ovladače nejpřirozenější, viz 2.5.3.

OvS nicméně offloaduje RTE flow pravidla pouze přes porty, které používá pro přenosy paketů. Z tohoto důvodu je nutné poskytnout RTE flow rozhraní i ovladači XL710 virtuálních funkcí `i40evf`. Ty jsou totiž v rámci navrženého akceleračního prototypu použity pro OvS porty.

Jelikož obě RTE flow rozhraní v těchto dvou DPDK ovladačích mají mnoho společného, byla vytvořena speciální složka `drivers/common/ipn3ke` pro soubory, které jsou sdíleny mezi ovladači `ipn3ke` a `i40evf`. Tato společná funkcionalita se týká zejména převodu RTE flow pravidel na vnitřní reprezentaci, která může být nahrána do firmware.

I když je možné některý kód používat oběma ovladači, jsou zde i problémy, které je nutné řešit odděleně. Hlavním z těchto problémů je nahrání pravidel do firmware. U ovladače `ipn3ke` se po převedení pravidla do vnitřní reprezentace může zavolat funkce pro nahrání pravidla do firmware pomocí `libp4dev`. Tuto možnost ale `i40evf` použít nemůže, jelikož nemá přímý přístup k `libp4dev` a dokonce ani k adresovému prostoru FPGA (jedná se



Obrázek 3.3: Navržená architektura pro RTE flow offload srze virtuální funkce

pouze o virtuální funkci). Zároveň není zaručeno, že `i40evf` běží ve stejné aplikaci jako `ipn3ke`, aby mohl zavolat funkce tohoto ovladače. Toto platí například pro výše zmíněný návrh prototypu, který je znázorněn na obrázku 3.2.

Proto bylo navrženo rozhraní, které využívá PF–VF zasílání zpráv. V takovém případě se pravidlo ve vnitřní reprezentaci ovladače zašle zprávou z VF do PF. U `i40e` fyzické funkce je již garantováno, že poběží ve stejné aplikaci jako `ipn3ke`, viz poslední odstavec kapitoly 2.5.3. Z `i40e` je tedy již možné zavolat funkci ovladače `ipn3ke` pro nahrání pravidla získaného ze zprávy do firmware.

Navržená architektura, která umožňuje RTE flow offload skrze `i40evf`, je znázorněna na obrázku 3.3. Zde je vidět, že VF aplikace může používat funkce pro přenosy paketů, ale nyní je schopna i RTE flow offloadu. Ten zajišťuje PF aplikace, kde ovladač `i40e` po přijetí zprávy zavolá funkci `ipn3ke`, která nahraje pravidlo do karty. PF aplikace je také schopna všech těchto funkcionalit (přenosu paketů, offload), nicméně v rámci akceleračního prototypu je použita pouze pro offload pravidel žádaných od VF aplikace. VF aplikaci v rámci prototypu představuje OvS a PF aplikaci představuje `vlan-mapper`.

Implementace společné funkcionality

Jak již bylo zmíněno, společná funkcionalita zahrnuje především převod pravidla ve formátu RTE flow do vnitřní reprezentace, která se dá snadno nahrát do firmware.

Pro tyto účely byla definována struktura `p4_flow`, která slouží pro ukládání informací o pravidlu potřebných pro `libp4dev` (parametry akcí, klíče a další). Zároveň je `p4_flow` navržena tak, aby co nejvíce usnadnila převod z RTE flow pravidla. Tento převod se dá rozdělit na čtyři hlavní části – inicializace `p4_flow`, přepis atributů, vzoru a akcí.

Během inicializace `p4_flow` se struktura naplní všemi potřebnými částmi pro nahrání pravidla. Nicméně všechny tyto části jsou vymaskovány tak, aby po nahrání neměly žádný

vliv. To znamená, že pravidlo popsané p4_flow by se do table0 v rámci teorie nahrát dalo, ale nad žádným paketem by žádnou změnu neprovedlo. V dalších částech převodu jsou pak určité vymaskované části p4_flow přepisovány daty z RTE flow pravidla. Tímto principem je "prázdné" pravidlo v p4_flow doplněno o informace z RTE flow atributů, vzoru a akcí.

Přepis atributů je jednoduchý, zde se pouze zkontroluje, zda neobsahují některé nepodporované části. Případně se zaznamená nějaký konkrétní atribut, který je potřeba v pozdějších fázích přepisu pravidla.

Přepis vzoru je složitější. Zde je nutné projít všechny položky (pattern items) a každou z nich zapsat do p4_flow. Pro každou podporovanou položku existuje specifická funkce pro zápis do p4_flow. Děje se tak, protože v RTE flow je každá položka specifikována pomocí unikátní struktury. Nejde k ní tedy přistupovat jako k poli bytů (to lze dělat maximálně na úrovni jednotlivých částí struktury). Pokud se ve vzoru vyskytuje nějaká nepodporovaná položka, převod je ukončen s chybou.

Před přepisem RTE flow akcí se provádí výběr základní akce z table0, které jsou popsány v kapitole 2.4. Základní akce je volena podle vstupních RTE flow akcí, tento výběr je znázorněn v tabulce 3.2.

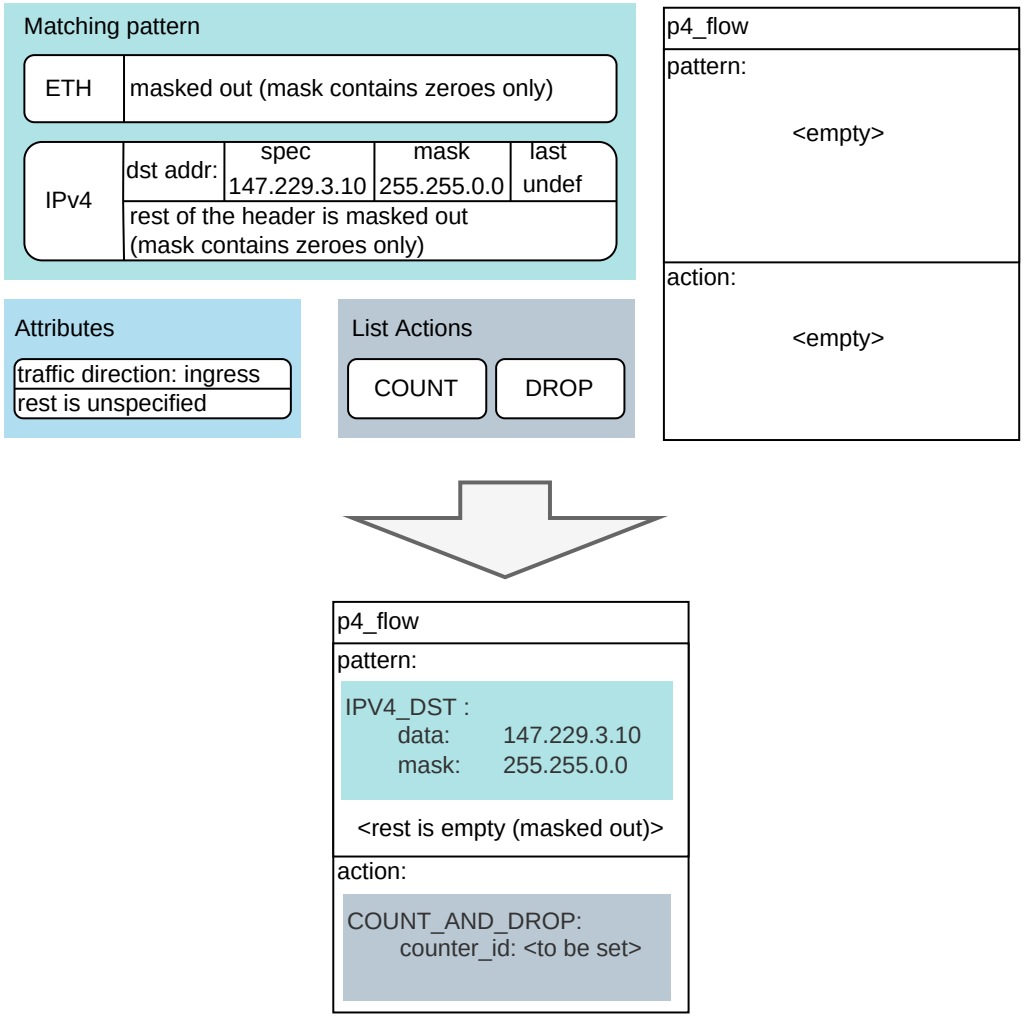
Vstupní RTE flow pravidla	Vybraná akce tabulky table0
DROP	Drop
COUNT a DROP	count_and_drop
Libovolná kombinace podporovaných akcí s výjimkou akce DROP	main_action

Tabulka 3.2: Výběr základní akce v table0

Průběh přepisu RTE flow akcí záleží na zvolené akci tabulky table0. V případě zvolených akcí Drop nebo count_and_drop již není nic více nutné do p4_flow přepisovat. V případě main_action je ovšem nutné projít RTE flow akce a přepsat do p4_flow důležité informace o této základní akci (parametry). Tento přepis probíhá podobně jako přepis vzoru. Prochází se zde jednotlivé RTE flow akce a pro každou existuje specifická funkce pro zapsání do p4_flow. Například u RTE flow akce SET_MAC_SRC se do p4_flow zapíše hodnota zdrojové MAC adresy, která se má v paketu změnit. Zároveň se do p4_flow zapíše bitová maska, která umožní změnu MAC adresy paketu. Princip těchto masek v main_action je popsán v kapitole 2.4. Pokud je během procházení nalezena nepodporovaná akce, převod je ukončen s chybou.

Na obrázku 3.4 je znázorněn převod RTE flow pravidla, které je příkladem v podkapitole 2.5.1. Je zde vidět prázdná struktura p4_flow, která je vyplněna daty z RTE flow akcí a vzoru. V atributech se nevyskytuje žádná informace, která je potřeba zapsat do p4_flow (směr ingress je jediný podporovaný, takže ho není potřeba zapisovat). Cílová IPv4 adresa ve vzoru je přepsána do odpovídající položky. Zbylé položky v p4_flow zůstávají vynulované, což znamená, že v rámci porovnávání nemají žádný vliv. Na základě RTE flow akcí COUNT a DROP je zvolena odpovídající akce, kterou table0 podporuje. Zde se vyskytuje jediný parametr counter_id, který je definován až v pozdějších fázích tvorby pravidla.

Jelikož je struktura p4_flow použita i při zasílání PF–VF zpráv pro offload, je vhodné ji detailněji popsat. Definici této struktury je možné vidět ve výpisu 3.1. Dále je zde možné vidět struktury pro akce main_action a count_and_drop_action, které obsahují místo pro jejich parametry. Struktura pro main_action zaznamenává ještě údaj o tom, zda byl v RTE flow pravidle přítomen atribut transfer, který mění význam parametrů. Ty mají



Obrázek 3.4: Příklad převodu RTE flow pravidla do struktury `p4_flow`

na starosti přesměrování paketu. Pro akci `Drop` žádná struktura neexistuje, jelikož žádné parametry neobsahuje.

Struktura `p4_flow_action` slouží pro reprezentaci libovolné z výše zmíněných akcí. Položka `generic_action` nese odkaz na jednu ze struktur pro akce, popřípadě je nastavena na `NULL` v případě akce `Drop`. Položka `type` určuje o jakou konkrétní akci se jedná, aby odkazovaná struktura mohla být správně interpretována.

Dále je možné na výpisu 3.1 vidět struktury pro klíče (`p4_flow_ternary_key`) a parametry akcí (`p4_flow_param`). Struktura `p4_flow_ternary_key` slouží pro uložení dat z RTE flow vzoru, která jsou následně poskytnuta k vytvoření klíčů (`p4_key_t`). Struktura `p4_flow_param` slouží pro uložení pomocných dat RTE flow akcí. Například u RTE flow akce `SET_TTL` by se ukládala data do dvou parametrů `main_action`. Jeden z nich by specifikoval novou hodnotu TTL a druhý by specifikoval masku, která určuje, zda se má nová TTL hodnota zapsat. Příklad těchto parametrů je uveden v tabulce 3.3. Kombinace takto zapsaných parametrů způsobí přepsání položky TTL na hodnotu 64. Hodnota maskovacího parametru 255 znamená, že se mají změnit všechny bity dané položky (v tomto případě TTL).

```
struct p4_flow_param {
        uint8_t size;
        uint8_t *data;
        const char *name;
};

struct p4_flow_ternary_key {
        const char *name;
        uint8_t size;
        uint8_t *data;
        uint8_t *mask;
};

struct p4_flow_count_and_drop_action {
        struct p4_flow_param counter_id;
};

struct p4_flow_main_action {
        struct p4_flow_param param[MAIN_ACTION_PARAM_NUM];
        bool attr_transfer;
};

struct p4_flow_action {
        enum p4_flow_action_type type;
        void *generic_action;
};

struct p4_flow {
        struct p4_flow_ternary_key pattern[TABLE0_KEY_NUM];
        struct p4_flow_action action;
};
```

Výpis 3.1: Definice p4_flow a jejích částí

Dále je možné si všimnout, že struktury pro klíče a parametry obsahují jen ukazatele na data, jelikož se ta data alokují dynamicky. Tento typ alokace byl vybrán, jelikož položky paketu mají různé velikosti a tento způsob dovoluje alokovat pouze tolik místa v paměti, kolik je potřeba. V případě statické alokace by pro každou položku muselo být alokováno tolik místa, kolik potřebuje největší položka.

Další důležitou strukturou, která byla definována, je `flow_p4_data`. Ta obsahuje dva identifikátory, první identifikuje pravidlo v hardware a druhý slouží pro identifikaci čítače, který se používá v rámci primitivní akce `count`. Tato struktura pak bude použita pro práci s pravidlem po jeho offloadování. Pomocí ní je možné pravidlo smazat, nebo vyčítat z jeho čítače.

`name`	`data`	`size` (velikost v bytech)
`new_ttl`	64	1
`ttl_mask`	255	1

Tabulka 3.3: Příklad parametrů, které způsobí změnu hodnoty TTL

Dále byla implementována funkce `ipn3ke_create_p4_flow`, která jako parametry přijme RTE flow pravidlo (atributy, vzor, akce) a prázdnou strukturu `p4_flow`. Tato funkce ve svém běhu vyplní strukturu `p4_flow` dle postupu, který je popsán výše.

Do souboru ipn3ke_p4dev.c byla přidána funkce pro nahrání p4_flow do firmware s názvem ipn3ke_p4dev_offload_p4_flow. Ta na základě struktury p4_flow vytvoří pravidlo (p4rule_t), které offloaduje do firmware. Toto pravidlo obsahuje specifikovanou akci z table0 (main_action, count_and_drop_action, nebo Drop) se specifikovanými parametry a klíči z p4_flow. Tato funkce v rámci svého běhu také inicializuje identifikátor pravidla v flow_p4_data pro budoucí práci s pravidlem. Jestliže však bude pravidlo obsahovat i primitivní akci count, musí být identifikátor čítače inicializován předem. K tomu slouží funkce ipn3ke_p4dev_set_optional_cnt, která zjistí, zda je identifikátor pro dané pravidlo potřeba a případně zaregistruje pomocí struktury counter_info v ipn3ke_p4dev nový identifikátor. Kromě tvorby pravidel je nutné tato pravidla i mazat, proto byla vytvořena funkce ipn3ke_p4dev_delete_p4_flow, která pravilo identifikované pomocí flow_p4_data smaže z firmware. Pro odregistraci čítače byla implementována funkce ipn3ke_p4dev_counter_remove. Všechny funkce popsané v tomto odstavci používají zámek v ipn3ke_p4dev, aby zajistili vzájemné vyloučení více jader při práci s **libp4dev** nebo strukturou counter_info.

Implementace RTE flow pro ipn3ke PMD

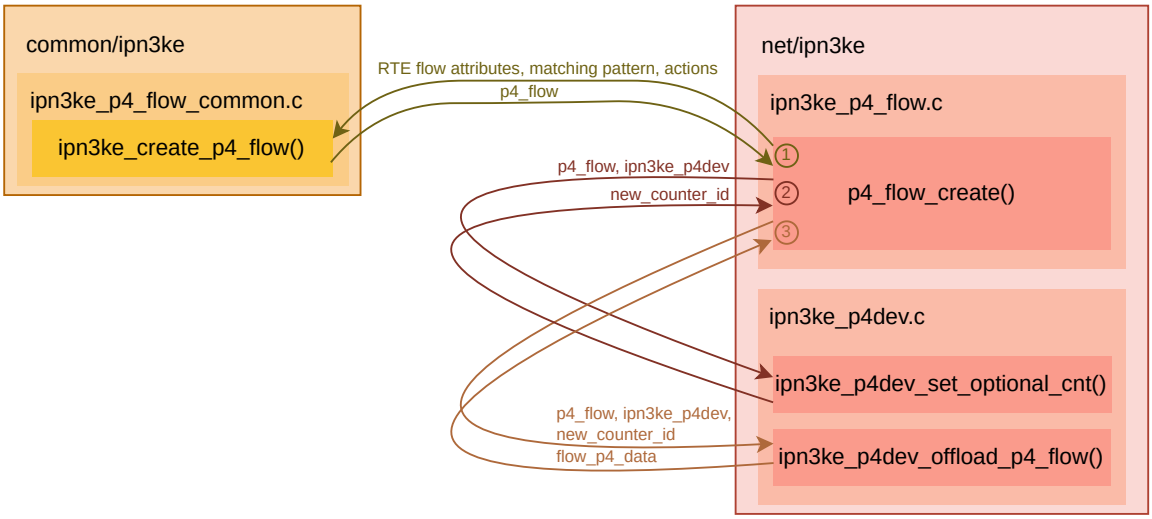
Výše uvedené rozhraní je schopné jak převést RTE flow pravidlo do interní reprezentace (struktura p4_flow), tak i tuto reprezentaci offloadovat do hardware, což z něj dělá základ pro tvorbu rozhraní RTE flow. Pro PMD **ipn3ke** byl vytvořen soubor ipn3ke_p4_flow.c, kde byly implementovány funkce pro callbacky z rte_flow_ops. Tyto funkce jsou uvedeny v tabulce 3.4. Zároveň byla definována samotná struktura rte_flow. Ta sestává ze struktury flow_p4_data a z odkazu na sebe sama, což umožní tvorbu jednosměrně vázaných seznamů.

Název callbacku	Název přiřazené funkce
create	p4_flow_create
destroy	p4_flow_destroy
query	p4_flow_query
flush	p4_flow_flush
validate	p4_flow_validate

Tabulka 3.4: RTE flow callbacky a jejich funkce v **ipn3ke**

Zjednodušený průběh funkce p4_flow_create je možné vidět na obrázku 3.5. Zde jsou vidět volání nejdůležitějších funkcí ve třech krocích. V prvním kroku se volá funkce ipn3ke_create_p4_flow, která na základě RTE flow atributů, vzoru a akcí vyplní strukturu p4_flow. V druhém kroku se inicializuje parametr new_counter_id. Tato inicializace se ovšem děje jen v případě, že dané pravidlo obsahuje akci COUNT. V třetím kroku se nahraje pravidlo v p4_flow do hardware funkcí ipn3ke_p4dev_offload_p4_flow. Ta posléze naplní strukturu flow_p4_data, díky které je možné v rámci aplikace provádět nad pravidlem další operace (callbacky query, destroy). Na obrázku je také možné si všimnout, že v rámci funkce p4_flow_create je použit i kód ze společného rozhraní (složka common/ipn3ke), které může být použito i ovladačem **i40evf**.

Ve funkci p4_flow_destroy se volá funkce ipn3ke_p4dev_delete_p4_flow pro odstranění pravidla specifikovaného flow_p4_data. Následně se volá funkce pro odregistraci čítače z firmware ipn3ke_p4dev_counter_remove a funkce pro jeho resetování, díky kterému muže být po odregistraci znova použit.



Obrázek 3.5: Volání nejdůležitějších funkcí v rámci callbacku `create` PMD **`ipn3ke`**

Před implementací funkce `p4_flow_flush` bylo nutné provést pár změn v již existujícím RTE flow rozhraní. Jako první byl přidán do struktury, která v ovladači **`ipn3ke`** představuje reprezentátory, ukazatel na `rte_flow`, který slouží jako odkaz na první prvek seznamu. Následně byla upravena funkce `p4_flow_create` tak, aby nově alokovanou strukturu připojila k seznamu. Podobně byla upravena funkce `p4_flow_destroy` tak, aby strukturu `rte_flow` před dealokací ze seznamu odpojila. Pro obě tyto akce byl použit zámek ve struktuře `ipn3ke_p4dev`, jelikož je možné, aby v rámci DPDK používalo RTE flow rozhraní více jader současně, což z tohoto seznamu činí kritickou sekci. Po těchto úpravách už bylo možné implementovat funkci `p4_flow_flush`. Ta volá v cyklu `p4_flow_destroy`, dokud není seznam `rte_flow` prázdný.

Dále byla implementována funkce `p4_flow_validate`. V tomto případě se pouze volá `ipn3ke_create_p4_flow`. Výsledná struktura `p4_flow` se uvolní a propaguje se návratová hodnota, která určuje, zda je dané pravidlo v rámci tohoto rozhraní proveditelné.

Jako poslední byla vytvořena funkce `p4_flow_query`, která se v rámci tohoto rozhraní dá kombinovat jen s pravidly s RTE flow akcí `COUNT`. Zde se děje pouze získání dat z čítače ve firmware, který je dostupný pomocí odkazu ve struktuře `flow_p4_data`. Kromě samotného vyčítání nabízí tato funkce také možnost data resetovat. Pro resetování a vyčítání dat se používá volání funkcí z knihovny **`libp4dev`**, u kterého se opět používá zámku ve struktuře `ipn3ke_p4dev` pro zajištění vzájemného vyloučení.

Implementace RTE flow pro `i40evf`

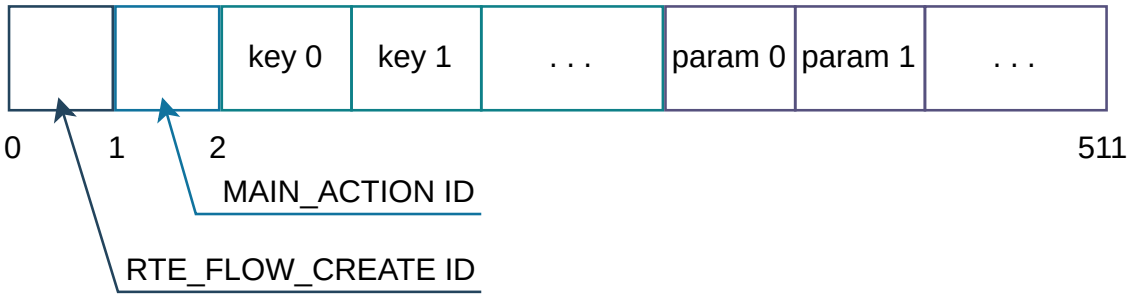
Jak již bylo zmíněno na začátku této kapitoly, rozhraní RTE flow ovladače **`i40evf`** je nutné implementovat skrze zasílání PF–VF zpráv, jelikož tento ovladač postrádá přístup k adresovému prostoru firmware i k samotné `libp4dev`.

Pro vytvoření RTE flow rozhraní pro ovladač **`i40evf`** bylo tedy nejprve nutné rozšířit **`i40e`** mechanismus PF – VF zasílání zpráv o nový typ zprávy, který označuje RTE flow operaci. Velikost zprávy byla definovaná na 512 bytů, což je dostatek i pro přenesení `p4_flow` struktury s datově nejobjemnější akcí `main_action`. Následně byl vytvořen jednotný formát těchto zpráv, aby bylo možné do zprávy zapsat potřebná data a následně je vyčíst zpět.

Prvním bytem zprávy je vždy identifikátor RTE flow callbacku. Zde se rozlišují 3 callbacky – `create`, `destroy` a `query`.

V případě `destroy` po identifikátoru callbacku následuje struktura `flow_p4_data`, tedy identifikátory pravidla a čítače.

U `query` následuje za identifikátorem callbacku boolovská hodnota, která definuje, zda se má čítač resetovat, nebo nikoliv. Za tímto políčkem je uložena struktura `flow_p4_data`.



Obrázek 3.6: Formát zprávy pro callback `create` a akci `main_action`

Zdaleka nejobjemnější zpráva je vysílána v rámci callbacku `create`. Zde je nutné uložit celou strukturu `p4_flow`. Po identifikátoru callbacku následuje identifikátor akce. Po něm jsou uloženy všechny klíče a po klíčích následují případné parametry. Příklad formátu pro akci `main_action` je možné vidět na obrázku 3.6. Funkce pro konverzi `p4_flow` do formátu zprávy a naopak byly implementovány do souboru `ipn3ke_p4_flow_common.c`, aby mohly být přístupné oběma ovladačům.

Po vytvoření formátu zpráv byla implementována funkce `send_flow_cmd`, která zašle z `i40e` VF zprávu do PF. Zpráva se zde zadává jako vstupní parametr a odpověď od fyzické funkce je výstupní parametr.

Následně byl vytvořen nový soubor `ipn3ke_p4_flow_vf.c` v ovladači `ipn3ke` pro funkce, které se budou volat jako odpověď na zprávy od VF. Funkce `p4_flow_vf_create` nejprve konvertuje zprávu z VF na strukturu `p4_flow`, poté zavolá funkci pro inicializaci `new_counter_id` `ipn3ke_p4dev_set_optional_cnt` a nakonec `p4_flow` offloaduje do firmware pomocí funkce `ipn3ke_p4dev_offload_p4_flow`. Jako odpověď na zprávu je do VF zaslána struktura `flow_p4_data`. Další funkcí je `p4_flow_vf_destroy`, která získá ze zprávy strukturu `flow_p4_data` a tu poté použije k resetování a odregistraci čítače a také pro smazání pravidla stejně jako callback `destroy` v ovladači `ipn3ke`. Jako odpověď tato funkce nezasílá nic, kromě své návratové hodnoty, která je zaslána i u všech ostatních funkcí. Ta se propaguje až do VF. Funkce `p4_flow_vf_query` získá ze zprávy údaj o tom, zda má být čítač resetován a také strukturu `flow_p4_data`. Poté vyčte hodnoty z čítače a případně jej resetuje, k čemuž je opět použito rozhraní ze souboru `ipn3ke_p4dev.c`. Vyčtené hodnoty funkce nakopíruje do odpovědi.

Tyto funkce (`p4_flow_vf_create`, `p4_flow_vf_destroy`, ...) musí být implementovány v **`ipn3ke`** ovladači, jelikož potřebují přistupovat k firmware přes rozhraní v souboru `ipn3ke_p4dev.c`. Zároveň je ale nutné volat tyto funkce z ovladače **`i40e`**, kde se reaguje na VF zprávy. Proto byla implementována funkce `ipn3ke_p4_flow_vf_process`, která příjme jako parametr zprávu z VF a podle prvního bytu zavolá odpovídající funkci (`p4_flow_vf_create`, `p4_flow_vf_destroy`, ...). Tato funkce je sice implementována v ovladači `ipn3ke`, ale během inicializačních rutin ovladačů se předá i do `i40e` zařízení, která ji potom mohou volat.

Dále byla rozšířena funkce `i40e_pf_host_handle_vf_msg` v souboru `i40e_pf.c` tak, aby byla schopna přijmout zprávu, která označuje RTE flow operaci. Pro tuto zprávu se volá funkce `ipn3ke_p4_flow_vf_process`, která zajistí realizaci zakódovaného pravidla.

S těmito úpravami ovladačů **`ipn3ke`** a **`i40e`** je možné implementovat RTE flow rozhraní pro **`i40evf`**. Pro tyto účely byl vytvořen soubor `i40e_p4_flow.c`, ve kterém byly implementovány funkce pro RTE flow callbacky. Tyto funkce je možné vidět v tabulce 3.5. Zároveň jsou zde vidět funkce, které se zavolají po zaslání zprávy do PF.

Název callbacku	Název přiřazené funkce	Funkce, která se volá z PF aplikace
`create`	`i40e_p4_flow_create`	`p4_flow_vf_create`
`destroy`	`i40e_p4_flow_destroy`	`p4_flow_vf_destroy`
`query`	`i40e_p4_flow_query`	`p4_flow_vf_query`
`flush`	`i40e_p4_flow_flush`	–
`validate`	`i40e_p4_flow_validate`	–

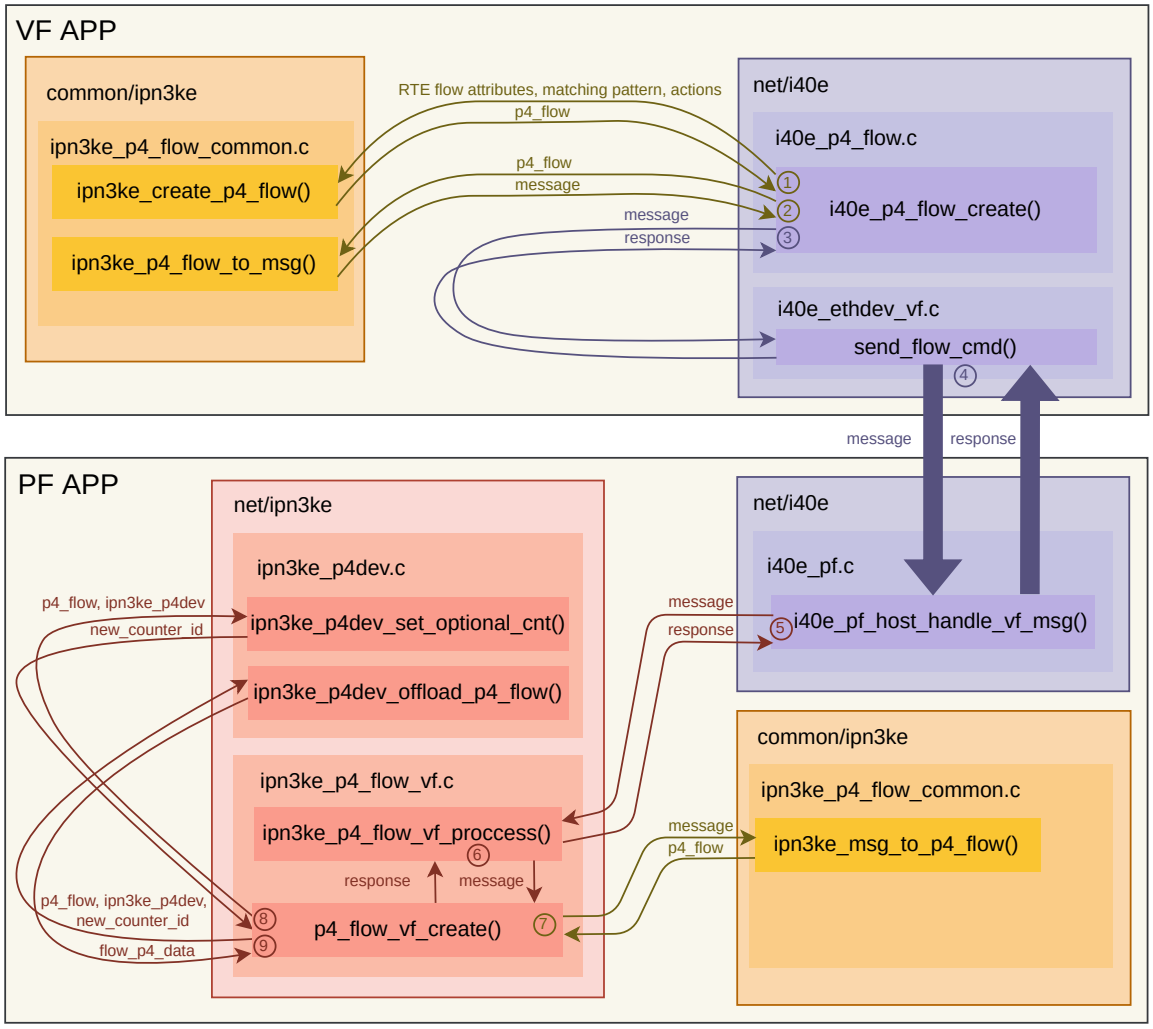
Tabulka 3.5: RTE flow callbacky a jejich funkce v **`i40evf`**

Ve funkci `i40e_p4_flow_create` se nejprve převedou RTE flow atributy, vzor a akce do `p4_flow` pomocí funkce `ipn3ke_create_p4_flow`. Následně se převede struktura `p4_flow` do formátu zprávy a zašle se pomocí funkce `send_flow_cmd` do PF. Z odpovědi na tuto zprávu je pak získána struktura `flow_p4_data`, která se vloží do `rte_flow`. Jako poslední se uloží `rte_flow` do seznamu, který existuje v rámci daného **`i40evf`** zařízení, díky kterému je možné implementovat funkci pro callback `flush`. Tento seznam je opět považován za kritickou sekci a při přístupu k němu je použito zámku typu `spinlock`.

Zjednodušený průběh této funkce je možné vidět na obrázku 3.7. Zde jde vidět chování funkce v jednotlivých bodech:

1. RTE Flow atributy, vzor a akce jsou převedeny na `p4_flow`.
2. Struktura `p4_flow` je převedena do pole bytů, které bude odesláno jako zpráva do PF.
3. Volání funkce `send_flow_cmd`, které je předána zpráva (pole bytů, ve kterém je zapsaná `p4_flow` a identifikátor callbacku `create`).
4. Ve volání funkce `send_flow_cmd` je zaslána zpráva do PF aplikace.
5. Příjem zprávy v PF aplikaci. Na základě typu zprávy (RTE flow offload) je volána funkce `ipn3ke_p4_flow_vf_process`.
6. Ve funkci `ipn3ke_p4_flow_vf_process` se vybere vhodná funkce na základě prvního bytu zprávy, který slouží jako identifikátor callbacku. Zavolá se vybraná funkce `p4_flow_vf_create`.
7. Ve funkci `p4_flow_vf_create` se nejprve převede zbytek zprávy do `p4_flow`.
8. Poté je případně zaregistrován čítač (pokud obsahuje pravidlo akci `count`)
9. Nakonec je pravidlo popsané `p4_flow` nahrané do firmware. Následně je získaná struktura `flow_p4_data` nahrána do odpovědi, která se propaguje zpět do funkce `i40e_pf_host_handle_vf_msg`, kde je odeslána zpět do VF. Poté je struktura

`flow_p4_data` uložena do `rte_flow`, která je připojena do seznamu a odkaz na ni je vrácen uživateli skrze funkci `rte_flow_create`.



Obrázek 3.7: Volání nejdůležitějších funkcí v rámci callbacku `create` PMD `i40evf`

Funkce `i40e_p4_flow_destroy` nakopíruje do zprávy `flow_p4_data` z `rte_flow`, následně ji zašle do PF a odstraní danou `rte_flow` z interního seznamu s použitím zámku.

Ve Funkci `i40e_p4_flow_query` se do zprávy nakopíruje hodnota, která udává, zda má dojít k resetu čítače. Dále se do zprávy nakopíruje `flow_p4_data` a následně se zpráva zašle do PF. Z odpovědi na tuto zprávu získá tato funkce údaje o paketech, které se shodovaly se vzorem pravidla, a předá je uživateli.

Zbylé funkce pro callbacky `validate` a `flush` jsou implementovány stejně jako v PMD `ipn3ke`. V rámci `validate` se volá funkce `ipn3ke_create_p4_flow`, čímž se zjistí, jestli je možné vytvořit `p4_flow`. Tím pádem se také zjistí, jestli je možné RTE flow pravidlo nahrát do hardware. V rámci `flush` se opět volá funkce pro `destroy` v cyklu dokud není seznam `rte_flow` prázdný.

3.1.4 Prostředí pro práci s prototypem

Po všech předchozích krocích implementace se prototyp stává plně funkčním, nicméně samotný prototyp je náročné používat, jelikož žádná práce s ním není automatizovaná. To znamená, že uživatel musí sám inicializovat kartu N3000, připravit stroj pro práci s DPDK, konfigurovat OvS, spouštět virtuální stroje pomocí QEMU a další. Z tohoto důvodu byly vytvořeny skripty v jazyce `bash`, které práci s prototypem usnadňují. Tyto skripty jsou umístěny ve složce `helper_scripts`.

Hlavní soubor, ve kterém je implementována většina funkcí, se jmenuje `common.sh`. V hlavičce souboru jsou definovány společné proměnné pro jednotlivé funkce, ty zahrnují jména pracovních složek, PCI identifikátory karty N3000 a VF token. Tento token je nutné předat všem aplikacím, které používají VFIO SR–IOV zařízení při startu, což zahrnuje OvS, QEMU a `vlan-mapper`. Funkce v tomto souboru se dají rozdělit do 3 základních skupin.

První z nich jsou funkce pro instalaci prostředí, které zahrnují překlad a instalaci OPAE, OvS, QEMU, DPDK a `libp4dev`. Tyto funkce je potřeba na daném stroji spustit pouze jednou.

Druhá skupina funkcí slouží pro nastavení prostředí pro spuštění prototypu. Na rozdíl od první skupiny, tyto funkce je potřeba spustit pokaždé, když je stroj znova zapnut. Tyto funkce zahrnují rezervaci `hugepages`, zapojení VFIO ovladače na kartu N3000 a tvorbu virtuálních funkcí.

Poslední skupinou jsou funkce, které se používají přímo za běhu prototypu. Zde se vyskytují wrappery pro DPDK aplikace a spouštění virtuálních strojů. Ty jednak zjednoduší zadávání mnoha parametrů a zároveň umožní pohodlnější předávání VF tokenu. Ten musí být stejný jak pro `vlan-mapper` (PF aplikace), tak pro virtuální stroje a OvS (VF aplikace). Dále jsou zde wrappery pro `ssh`, které při spuštění připojí uživatele na daný virtuální stroj. Zbylé funkce slouží pro konfiguraci OvS, což zahrnuje resetování OvS databáze, vytvoření OvS `bridge` a připojení portů, které používají XL710 virtuální funkce.

3.2 Testování

Vývoj tohoto prototypu byl veden iteračním způsobem, ve kterém se po implementaci jedné z částí prototypu prováděly testy dané části a až poté se pokračovalo v další implementaci. Testování tedy probíhalo ve třech hlavních krocích.

Prvním krokem je testování funkce knihovny `libp4dev` v PMD `ipn3ke` 3.2.1. Druhým krokem je testování funkčního SR–IOV síťového rozhraní 3.2.2. Třetím a posledním krokem je testování implementovaného RTE flow rozhraní 3.2.3.

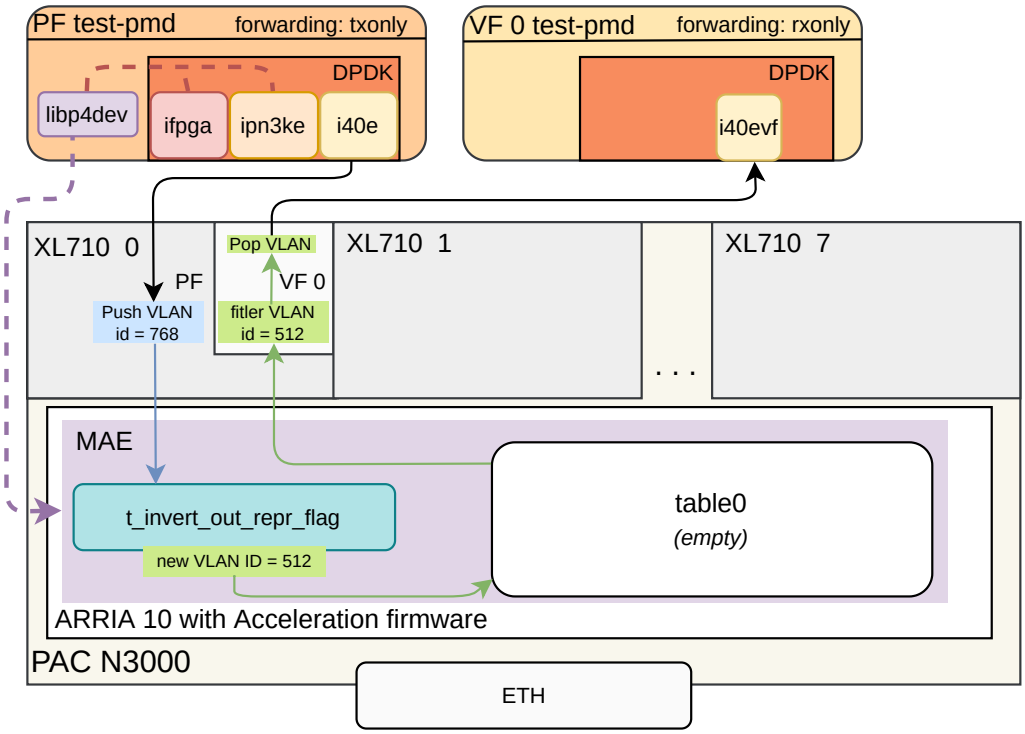
Všechny tyto testy probíhaly na serveru `oliver`, na který byla připojena karta PAC N3000. Připojená karta disponuje hardwarovou konfigurací 8 × 10GbE, což znamená, že oba čipy XL710 obsahují čtyři kanály o rychlosti 10 Gb/s. Každý z těchto kanálů je v software reprezentován jedním PCI zařízením, což znamená, že PAC N3000 je viděna jako 9 zařízení (8 × XL710 + 1 × FPGA Arria 10).

3.2.1 Testování funkce knihovny `libp4dev` v PMD `ipn3ke`

První krok testování se týká portace knihovny `libp4dev` do DPDK ovladače `ipn3ke`, jejíž návrh a implementace je popsán v podkapitole 3.1.1. Zde je nutné ověřit funkčnost `libp4dev`, která přistupuje do hardware skrze přidělené callbacky.

Jako první byla ověřena inicializace a povolení chodu `match-action engine` pomocí `libp4dev`. Díky těmto operacím je možné kartu PAC N3000, ve které je nahrán akcelerační firmware, používat pro síťový provoz. Ověření spočívalo v zaslání paketů do karty a jejich opětovném přijetí. Tento test je znázorněn na obrázku 3.8.

Zde je vidět, že první XL710 čip (index 0) je pomocí SR–IOV rozdělen na jednu fyzickou a jednu virtuální funkci. Každá z těchto hardwarových funkcí je předána do DPDK, konkrétně do testovacího DPDK nástroje `test-pmd`. Pomocí něj jsou nahrána do XL710 VLAN pravidla, která způsobují filtrování, odstraňování a přidávání VLAN hlavičky.



Obrázek 3.8: Test správné funkce `match-action engine` (MAE) po inicializaci

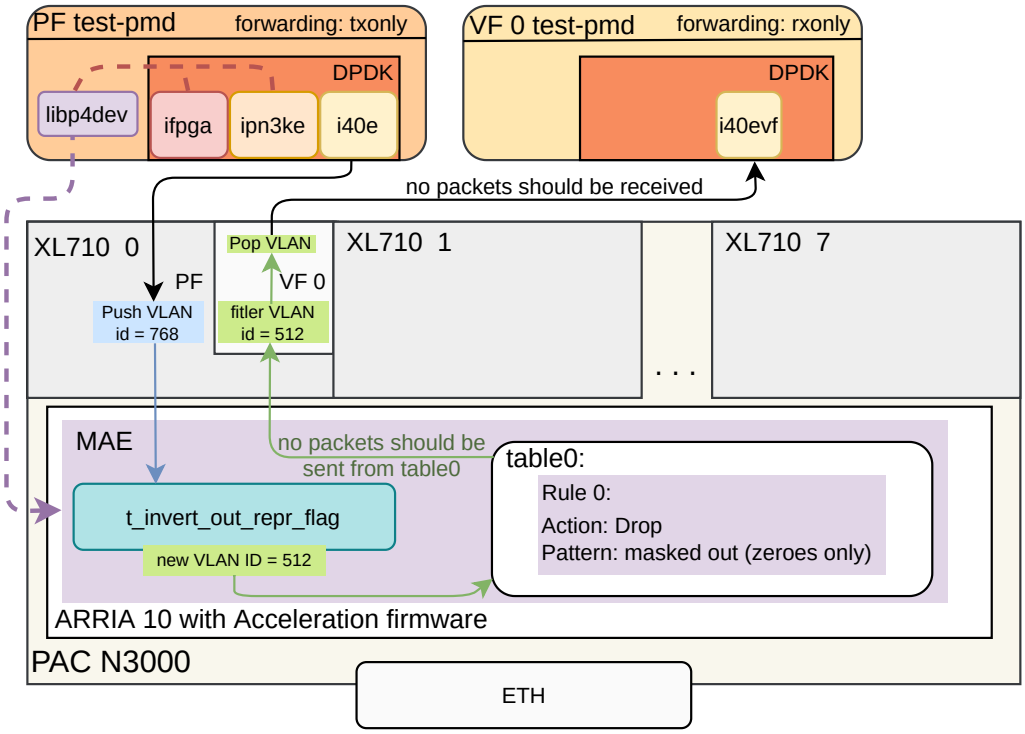
Dále je na obrázku 3.8 možné vidět, že instance `test-pmd`, která používá fyzickou funkci je nastavena v režimu `txonly`, ve kterém pouze vysílá pakety. Tyto pakety pokračují směrem do XL710, kde jim jsou přidány VLAN hlavičky s VID 768. Tyto pakety se následně zašlou do `match-action engine` (MAE), kde se jim přepíše hodnota VID na 512 v tabulce `t_invert_out_repr_flag`. Následně pakety pokračují do tabulky `table0`, ve které nejsou změněny, jelikož do ní není nahráno žádné pravidlo. Poté jsou zaslány zpět do XL710 čipu. Zde se na základě hlavičky VLAN a jeho identifikátoru zašlou do virtuální funkce a daná hlavička je jim odebrána. Nakonec jsou pakety přijaty instancí `test-pmd`, která virtuální funkci využívá.

Díky tomuto testu bylo ověřeno, že se dá karta PAC N3000 s akceleračním firmware použít pro přeposílání paketů v rámci DPDK.

Druhý test probíhal stejným způsobem až na fakt, že při inicializaci karty bylo do `table0` nahráno jednoduché pravidlo. V rámci tohoto testu byla vyzkoušena různá jednoduchá pravidla. Tato pravidla typicky disponovala prázdnou `pattern`, což znamená, že všechny klíče pro porovnávání byly vymaskovány tak, aby se pravidla aplikovala na každý

paket. Akce těchto pravidel byly nastaveny například na `Drop`, nebo `main_action`, kde bylo nastavené přepisování MAC adresy.

Po přijetí paketů v instanci **`test-pmd`**, která využívá virtuální funkci bylo zkoumáno, zda na pakety bylo aplikováno nahrané pravidlo. Například v případě změny MAC adresy byla tato položka paketu kontrolována. Na obrázku 3.9 je znázorněn testovací běh, kde bylo při inicializaci nahrané pravidlo, které zahazuje veškeré pakety. Správná funkce **`match-action engine`** se zde prokáže tím, že v instanci **`test-pmd`**, která využívá virtuální funkci, není přijat žádný paket (všechny jsou zahozeny).



Obrázek 3.9: Test správné funkce MAE po nahrání testovacího pravidla

Oba tyto testy ověřují schopnost **`libp4dev`** pracovat s **`match-action engine`**, což poskytuje možnost implementace dalších částí prototypu.

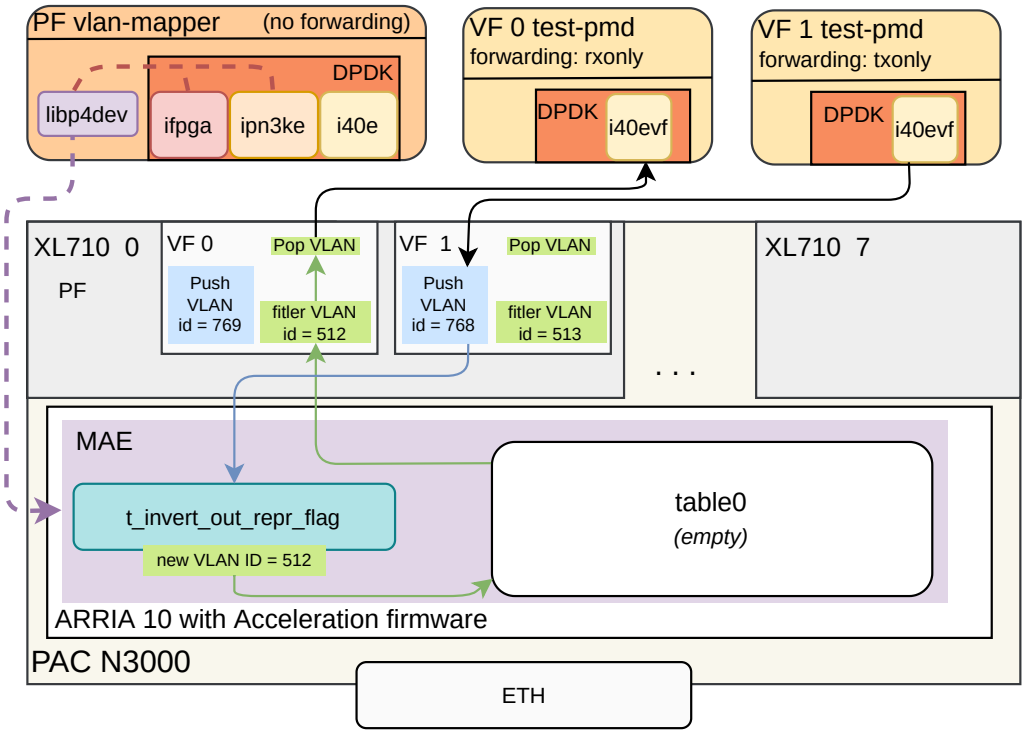
3.2.2 Testování funkčního SR–IOV síťového rozhraní

Druhý krok testování se týká funkčního SR–IOV síťového rozhraní, které je zaručeno aplikací **`vlan-mapper`**. Návrh a implementace testovaného rozhraní jsou popsány v podkapitole 3.1.2. Toto rozhraní bylo nejprve testováno pomocí **`test-pmd`**, následně pomocí dvou virtuálních strojů a nakonec pomocí samotného OvS.

Testování pomocí `test-pmd`

První test v tomto kroku je zaměřen na ověření správného VLAN mapování, které generuje aplikace **`vlan-mapper`**. Schéma tohoto testu je znázorněno na obrázku 3.10. Zde je vidět aplikace **`vlan-mapper`**, která používá fyzickou funkci a VLAN pravidla v XL710, která byla touto aplikací nahrána. V tomto testu existují dvě virtuální funkce, které jsou pomocí

těchto VLAN pravidel na sebe namapovány. Každou virtuální funkci používá jedna instance `test-pmd`, přičemž jedna z nich pakety vysílá a druhá je přijímá. V další části testu jsou prohozeny režimy zpracování paketů těchto `test-pmd` instancí, aby se ověřilo, že přenosy paketů fungují oběma směry.



Obrázek 3.10: Test VLAN mapování ve směru VF 1 → VF 0

Testování pomocí virtuálních strojů

Po ověření funkčnosti mapování na testovacích paketech je vhodné jej otestovat i pomocí běžného síťového provozu. Na tuto problematiku je zaměřen druhý test, který využívá virtuálních strojů pro generování síťového provozu.

Fyzická funkce je v tomto testu opět využívaná aplikací `vlan-mapper`. V testu se opět vyskytují dvě virtuální funkce. Každá z těchto virtuálních funkcí je připojena na jeden virtuální stroj. Ve virtuálních strojích běží Linux kernel ovladač `iavf`, který je použit pro vytvoření klasického síťového rozhraní. To je možné využívat síťovými aplikacemi, jako je například `ping`, `ssh` nebo `tcpreplay`.

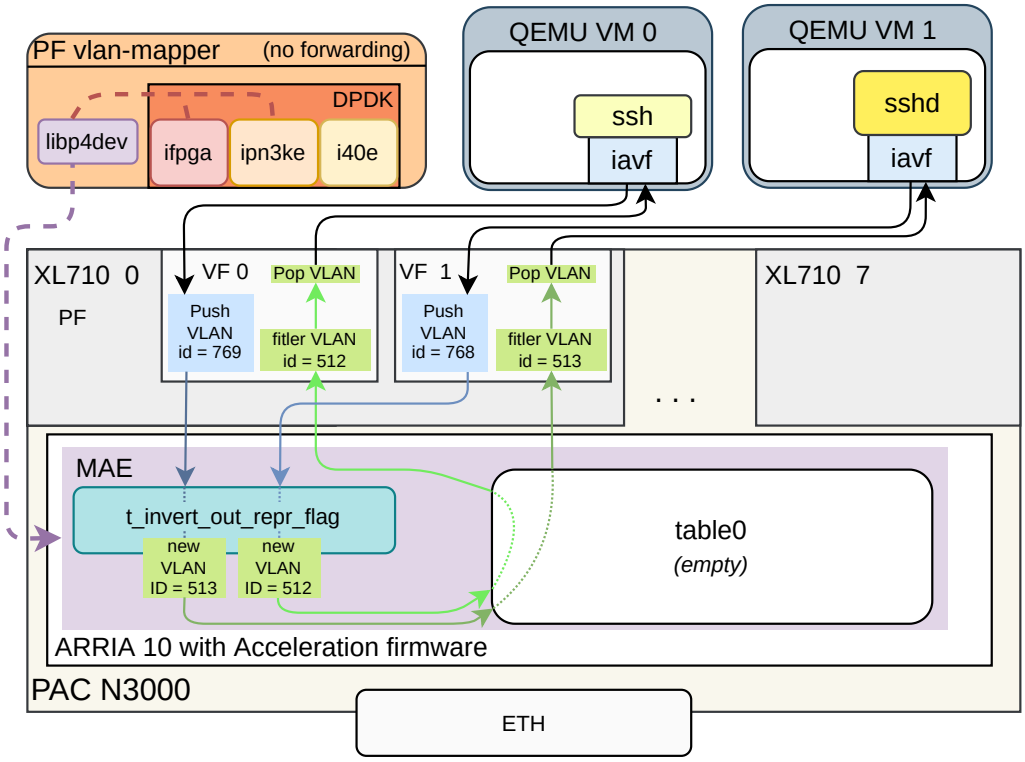
VLAN mapování bylo v tomto testu nejprve ověřeno příkazy `tcpreplay` a `tcpdump`, pomocí kterých je možné zaslat paket ve formátu `pcap` a následně ho znovu přijmout. Tyto příkazy sice plně nedokazují schopnost síťového provozu, ale dá se pomocí nich ověřit, že je možné mezi virtuálními stroji posílat pakety.

V dalším kroku bylo rozhraní ověřeno příkazy `ping` a `ssh`, díky kterým je možné potvrdit, že je prototyp schopen běžného síťového provozu.

Test pomocí `ssh` spojení je znázorněn na obrázku 3.11. Zde je vidět stejné zapojení i mapování jako v předešlém testu, jen jsou na místo `test-pmd` aplikací připojeny k VF0 a VF1 virtuální stroje. Také je oproti předchozímu testu vidět zasílání paketů oběma směry,

jelikož se jedná o `ssh` spojení (které oběma směry komunikuje). Ve virtuálním stroji VM1 běží `ssh` démon a ve VM2 je spuštěn příkaz `ssh`, který naváže spojení s VM1. Během tohoto testu bylo vyzkoušeno ustavení `ssh` spojení. Následně bylo zasláno několik stovek MB pomocí kombinace příkazů `ssh` a `dd` z jednoho virtuálního stroje na druhý.

Vytvoření tohoto spojení potvrzuje, že je možné posílat pakety oběma směry současně. Zároveň je také potvrzeno, že je možné toto rozhraní používat pro běžný síťový provoz (všechny předchozí testy používaly jen testovací pakety).



Obrázek 3.11: Test VLAN mapování pomocí virtuálních strojů a spojení `ssh`

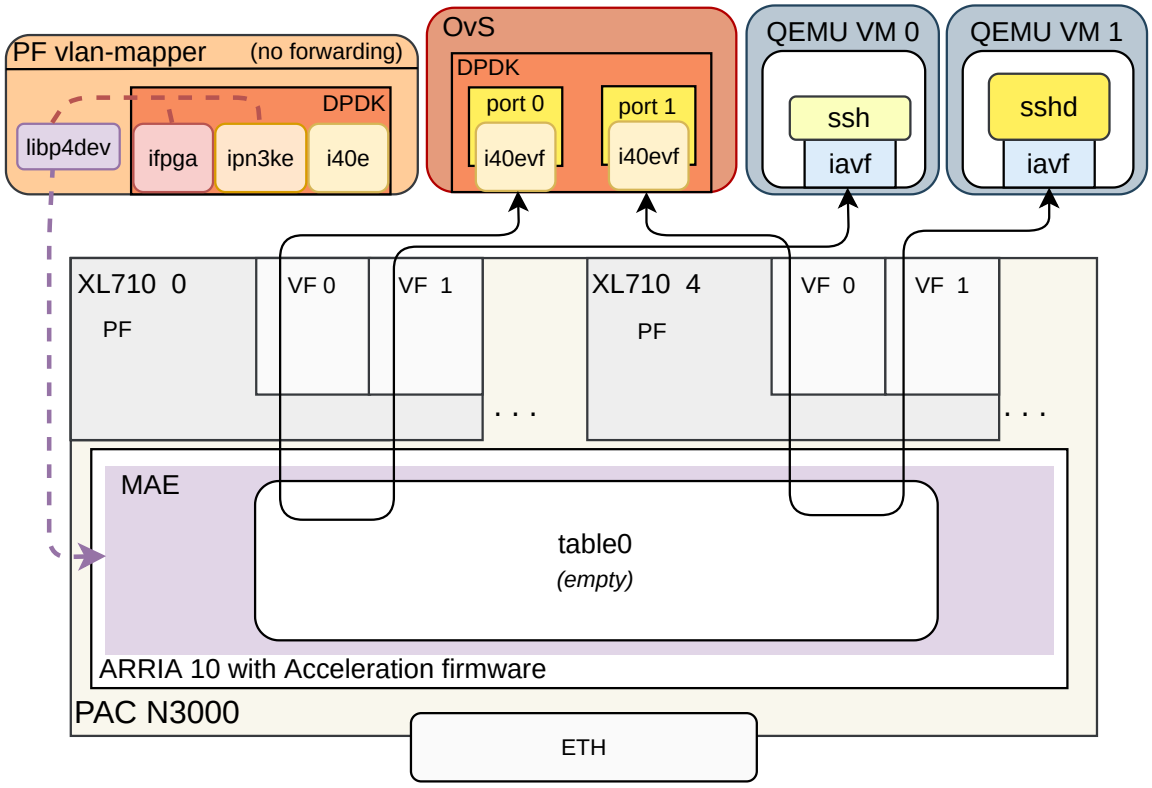
Testování na OvS

Posledním testem v rámci tohoto kroku je ověření funkce rozhraní na samotném OvS. V tomto testu jsou potřeba čtyři virtuální funkce. Dvě z nich slouží portům OvS a zbylé dvě jsou přiděleny virtuálním strojům jako v předchozím případě. Virtuální funkce musí být přiděleny tak, aby na sebe byly namapovány porty OvS s virtuálními stroji.

Tento test je znázorněn na obrázku 3.12. Zde jsou namapované VF0 s VF1 u obou použitých XL710 zařízení. Samotné mapování je na tomto obrázku pouze naznačeno pomocí šipek, jelikož jeho plné zobrazení jako v předchozím obrázku by již bylo vzhledem k počtu virtuálním funkcím nepřehledné. Skrytá zůstává i tabulka `t_invert_out_repr_flag`, která by také v novém schématu působila nepřehledně. Nicméně toto mapování zůstává pořád stejné v obou XL710 zařízeních. Jediným rozdílem jsou odlišné hodnoty VLAN id.

Dále je na tomto obrázku možné vidět OvS, které skrze DPDK používá virtuální funkce XL710. Tyto funkce jsou namapovány na virtuální funkce, které jsou připojeny k virtuálním strojům. Díky tomuto mapování jsou virtuální stroje připojeny k OvS a je možné otestovat síťový provoz mezi těmito virtuálními stroji. Testy komunikace virtuálních strojů probíhaly

stejně jako v předchozím případě. Na obrázku 3.12, je vidět `ssh` spojení mezi virtuálními stroji, jehož funkce potvrzuje korektní VLAN mapování a funkci OvS (přepínání paketů). Během tohoto testu bylo vyzkoušeno ustavení `ssh` spojení a následné zaslání několika stovek MB přes `ssh` jako v předchozím případě.



Obrázek 3.12: Test VLAN mapování pomocí OvS (mapování naznačeno pouze šipkami)

3.2.3 Testování RTE flow

V třetím a posledním kroku je testováno rozhraní RTE flow, které je navržené a implementované v podkapitole 3.1.3. Nejprve bylo toto rozhraní testováno pro ovladač `ipn3ke` pomocí nástroje `test-pmd`, aby bylo ověřeno nahrávání složitějších pravidel do `table0`. Následně bylo testováno rozhraní RTE flow pro ovladač `i40evf`. To bylo testováno nejprve pomocí nástroje `test-pmd`, aby bylo ověřeno zasílání `p4_flow` a dalších informací skrze PF–VF zprávy. V posledním kroku je prototyp otestován jako celek na OvS, kde je povolen hardwarový offload, což znamená použití RTE flow rozhraní ovladače `i40evf` v rámci běžného síťového provozu.

Testování RTE flow rozhraní ovladače `ipn3ke`

Pro testování RTE flow ovladače `ipn3ke` byly použity dva virtuální stroje, které byly připojeny na dvě XL710 virtuální funkce. Zbytek karty byl předán nástroji `test-pmd`. Schéma tohoto testu by vypadalo stejně jako na obrázku 3.11, jen s rozdílem, že místo aplikace `vlan-mapper` je spuštěn příkaz `test-pmd` a ve virtuálních strojích jsou spuštěny příkazy `tcpdump` a `tcpreplay`. Na začátku testování bylo provedeno stejné VLAN mapování po-

mocí `test-pmd`, které by provedl `vlan-mapper`. Jediný rozdíl mezi těmito mapováními je počet příkazů potřebných pro jejich provedení. Zatímco u aplikace `vlan-mapper` stačí provést příkaz `vlan-map`, u `testpmd` bylo zapotřebí zadat všechny příkazy pro VLAN offload ručně. Testy probíhaly následujícím způsobem:

1. Do `table0` byl nahrán soubor testovacích RTE flow pravidel skrze `test-pmd`.
2. Poté byly z jednoho virtuálního stroje zaslány testovací pakety ve formátu `pcap` pomocí příkazu `tcpreplay` a byly přijaty na druhém virtuálním stroji pomocí příkazu `tcpdump`
3. Pak byly zkoumány účinky, které mají na přijaté pakety nahraná pravidla.
4. Nakonec byla testovací pravidla z `table0` smazána.

Testovací RTE flow pravidla se zde dají rozčlenit do tří základních skupin – pravidla testující vzor, pravidla testující akce a náhodná pravidla.

Pravidla testující vzor sestávají z odlišných položek vzoru ale stejných akcí, aby se dalo jednoduše rozlišit, jestli se akce na paket aplikovaly (`match`). Pro tyto účely byla použita kombinace RTE flow akcí `COUNT` a `DROP`. Aplikace těchto akcí na pakety se dá velmi dobře detekovat, jelikož pakety vyhovující vzoru jsou zahozeny. Počet takto zahozených paketů se dá díky akci `COUNT` zpětně vyčíst pomocí funkce `rte_flow_query`.

Pravidla testující akce sestávají z prázdného vzoru, což znamená, že se aplikují na jakýkoliv paket. To usnadňuje zkoumání účinků daných akcí na pakety. Toto zkoumání se u každé akce liší. V případě akcí pro přepis položky paketu, jako je například `SET_IPV4_SRC`, je nutné zobrazit přijatý paket ve formátu `pcap` do čitelné formy (například pomocí `wireshark` aplikace). Poté je možné zjistit jestli se dané políčko paketu změnilo. V případě akce `PORT_ID` se zkoumá, zda byl paket přijat na specifikovaném portu. Naopak u akce `DROP` se ověřuje, zda paket nikde přijat nebyl (byl zahozen). U akce `COUNT` se ověřuje, zda se vyčtený počet paketů shoduje s počtem odeslaných paketů. Zároveň se také ověřuje, zda funguje reset čítače ať už na požádání (`rte_flow_query`) nebo při jeho odregistraci (`rte_flow_destroy`).

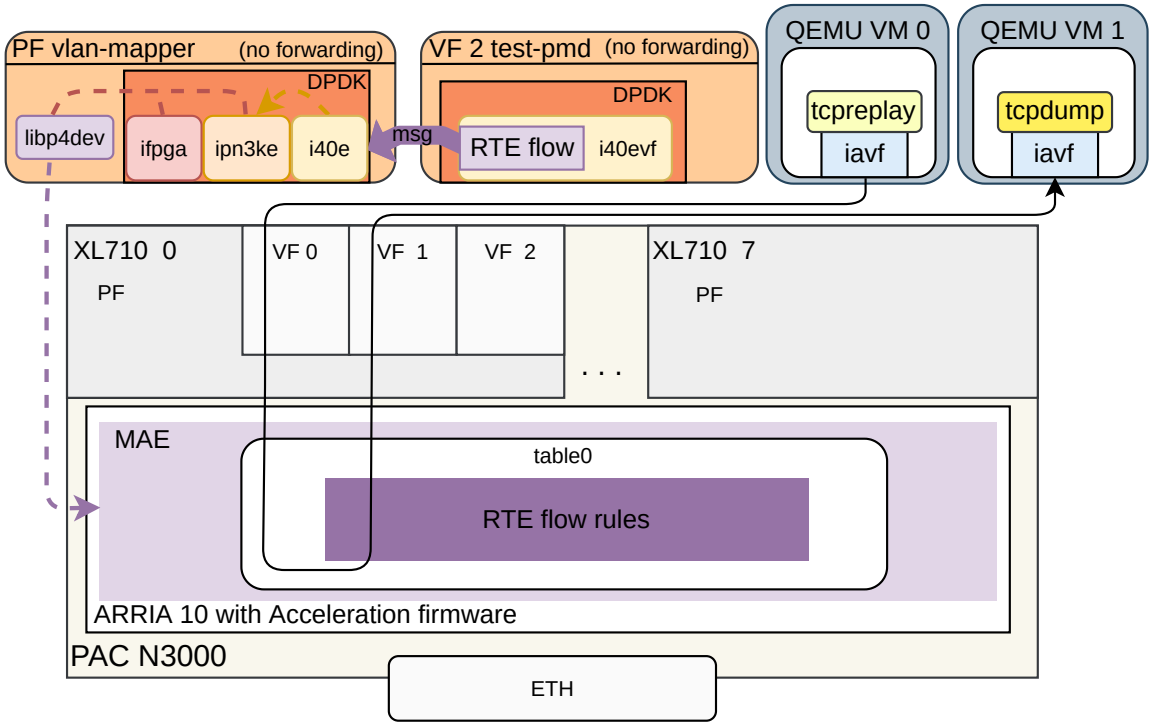
U náhodných pravidel jsou vybrány různé vzory i různé akce náhodně. Tato pravidla doplňují sadu testovacích pravidel a jsou prováděna pro kontrolu, zda dané akce fungují v kombinaci s danými vzory a naopak.

Testování RTE flow rozhraní ovladače `i40evf`

Testování RTE flow u ovladače `i40evf` probíhalo stejně jako u `ipn3ke`. Jediný rozdíl tvořily jednotlivé aplikace použité pro testování. V tomto případě zde existovaly tři XL710 virtuální funkce. Dvě z nich byly připojeny ke dvěma virtuálním strojům a třetí byla připojena k nástroji `test-pmd`. Zbytek karty byl připojen k aplikaci `vlan-mapper`.

Díky tomuto výběru použitých aplikací je možné otestovat přenos zpráv z `test-pmd` (VF2) do `vlan-mapper` (PF) za účelem RTE flow offloadu. Jako aplikaci, která používá PF, byl vybrán `vlan-mapper`, jelikož usnadňuje VLAN mapování (v `test-pmd` se musí všechna pravidla zadávat ručně).

Toto testování je znázorněno na obrázku 3.13. Zde je vidět přenos RTE flow pravidla skrze zprávu do PF aplikace (`vlan-mapper`) odkud je pravidlo nahráno do `table0` pomocí `libp4dev`. Práce s VLAN tagy pro účely mapování je i v tomto obrázku skryta.



Obrázek 3.13: Testování RTE flow rozhraní PMD `i40evf`

Postup u testů probíhal stejně jako u ovladače `ipn3ke`. Změnila se ovšem sada testovacích pravidel. Hlavním cílem testování `ipn3ke` RTE flow rozhraní je ověření, zda fungují všechny akce a klíče (položky vzoru). V tomto testování (`i40evf`) je hlavním cílem ověření, zda se zprávy s žádostí o RTE flow offload z VF do PF posílají korektně. Proto byly testovány různé kombinace RTE flow akcí, které mění výběr základní akce v `table0` (`main_action`, `count_and_drop` a `Drop`), což mění strukturu posílané zprávy. Dále se testování zaměřilo na RTE flow akci `COUNT` a vyčítání hodnot z čítače, který je registrován v rámci této akce. Tyto hodnoty jsou totiž také přenášeny pomocí zpráv.

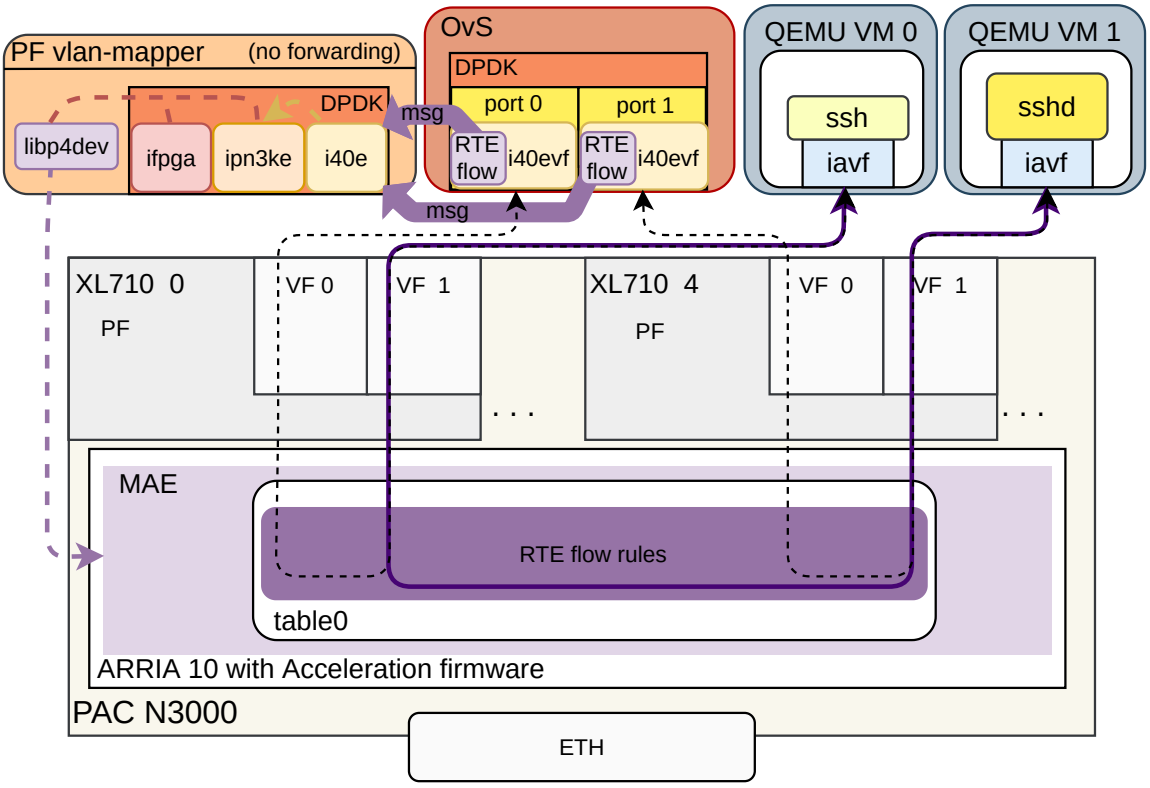
Testování hardwarového offloadu v OvS

V posledním kroku testování RTE flow byla ověřena funkce implementovaného rozhraní na OvS. V tomto případě je opět použito rozhraní ovladače `i40evf`, jelikož zařízení toho ovladače má OvS připojeno na své porty.

Tyto testy se velmi podobají předchozím testům s OvS, které jsou znázorněny na obrázku 3.12. Rozdíl je zde jeden, nicméně velmi zásadní – v OvS je povolen hardware offload. To znamená, že v případě potřeby OvS může přenést některé ze svých klasifikačních pravidel skrze RTE flow na úroveň síťové karty.

Na obrázku 3.14 je vidět schéma testů hardwarového offloadu OvS. Na tomto obrázku je možné si všimnout, že OvS přes své porty offloaduje svá klasifikační pravidla do firmware. Tento offload je realizován skrze implementované RTE flow rozhraní, což znamená, že jsou daná pravidla převedena do formátu zprávy a zaslána do PF aplikace (`vlan-mapper`). Teprve zde jsou nahrána pravidla do karty pomocí `libp4dev`.

Nahraná RTE flow pravidla v tomto případě způsobují odklon toku paketů od OvS přímo do cílových virtuálních strojů. Díky těmto pravidlům jsou pakety přímo zasílány do



Obrázek 3.14: Testování RTE flow rozhraní na OvS

virtuálních strojů a tím pádem nezatěžují OvS, což by se mělo projevit vyšší propustností a menším zatížením CPU. Podrobnější výsledky jsou rozebrány v následující kapitole 3.3.

3.3 Měření

Měření bylo provedeno ve dvou topologiích – PV (Physical interface to Virtual Interface) a PP (Physical interface to Physical interface). U těchto měření byl kladen důraz na propustnost.

Spotřeba CPU zde nebyla přímo měřena jelikož počet přepínaných paketů za jednotku času v případě DPDK verze OvS nemá na spotřebu procesoru žádný vliv. Tento jev je způsoben `polling` režimem, ve kterém DPDK ovladače fungují. V praxi to znamená, že jádra, která by v případě velkého síťového provozu byla zaměstnána přepínáním paketů, jsou použita pro `polling`. Ve výsledku je pak spotřeba CPU u OvS–DPDK stejná jak při malém, tak při velkém síťovém provozu.

Nicméně i tak je možné u OvS–DPDK zajistit, aby byla spotřeba CPU snížena. Samotná spotřeba je určena již při inicializaci OvS, kdy se DPDK předávají logická jádra. Optimální propustnosti dosahuje OvS–DPDK pokud je mu předáno alespoň tolik jader, kolik má funkčních portů. Pokud jsou některá z pravidel OvS přenesena na úroveň chytré síťové karty, je možné, aby byla dosažena stejná, nebo větší propustnost i za použití méně jader. Tento princip úspory CPU je měřen v rámci PV testování v podkapitole 3.3.2.

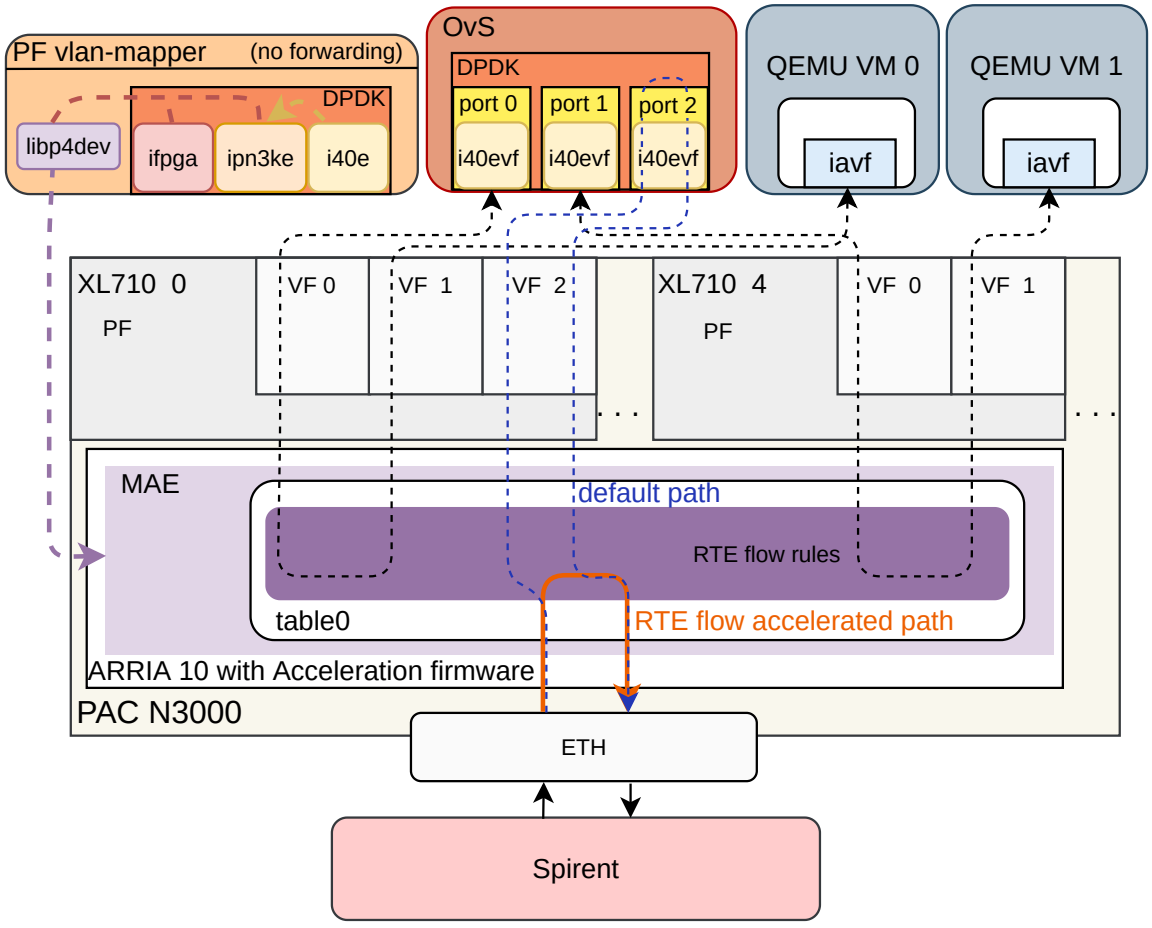
Měření probíhalo na stroji `sevar`, který obsahuje 8 procesorových jader Intel Xeon E5-2630 o frekvenci 2.40 GHz a 64 GiB operační paměti. Tento server je osazen síťovou kartou PAC N3000 s konfigurací 8 × 10 GbE. Na jeden z ethernetových portů byl připojen testovací

nástroj `spirent`, respektive jeho port o rychlosti 10 Gb. Tento nástroj je schopen generovat síťový provoz a měřit vstupní a výstupní rychlost.

U obou měření bylo použito OvS–DPDK, které disponuje třemi porty. Dva tyto porty byly připojeny k virtuálním strojům a poslední z nich byl použit pro připojení k ethernetovému portu (a tedy k přístroji `spirent`). Pro virtuální stroje i porty OvS byly použity XL710 virtuální funkce. Mapování těchto funkcí bylo provedeno pomocí aplikace `vlan-mapper`.

3.3.1 PP Topologie

Měření v rámci PP topologie spočívalo v přenosu paketů z fyzického portu (ethernet) do OvS. Odtud byly pakety zaslány zpět na fyzický port, kde byla měřena rychlost, ve které přicházely pakety z OvS. Pro tyto účely bylo pomocí příkazu `ovs-ofctl` přidáno pravidlo do OvS, které vynutilo přijaté pakety zaslat zpět do Ethernetového portu.



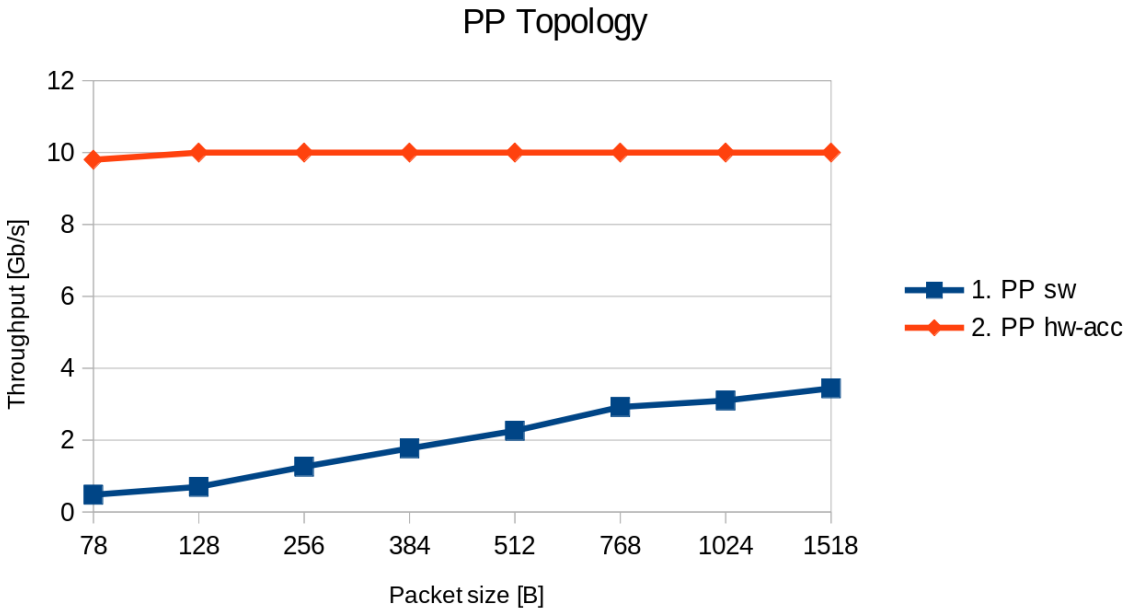
Obrázek 3.15: Schéma PP topologie u měření

Toto měření bylo provedeno ve dvou bězích, v prvním běhu byla v OvS vypnuta hardwarová akcelerace a v druhém zapnuta. V obou případech byla do DPDK přidělena 3 jádra procesoru, což zajišťuje optimální výkon softwarového přepínání paketů.

Na obrázku 3.15 je znázorněno schéma měření v rámci PP topologie s hardwarově akcelerovaným OvS. Zde jsou vidět nahraná RTE flow pravidla v tabulce `table0`, která

způsobují odklon přijatých paketů zpět do ethernetového portu (cesta paketů je znázorněna oranžově).

Dále je na obrázku modře naznačena cesta, kterou by pakety proudily, pokud by byla tabulka `table0` prázdná. Taková cesta byla použita v běhu OvS, kde je vypnuta hardwarová akcelerace.



Obrázek 3.16: Výsledky měření u PP topologie

V grafu na obrázku 3.16 jsou vidět naměřené propustnosti variant OvS s hardwarovým offloadem a bez něj. Zde je vidět, že i když se propustnost u OvS bez offloadu zvyšuje s velikostí paketu, i tak tato varianta nedosahuje propustnosti, kterou nabízí OvS s hardwarovým offloadem. Hlavním úzkým hrdlem je v tomto případě práce s pakety v software. Jelikož je hardwarově akcelerovaný Open vSwitch schopen přenést tuto práci na úroveň síťové karty, je možné dosáhnout plné propustnosti.

3.3.2 PV Topologie

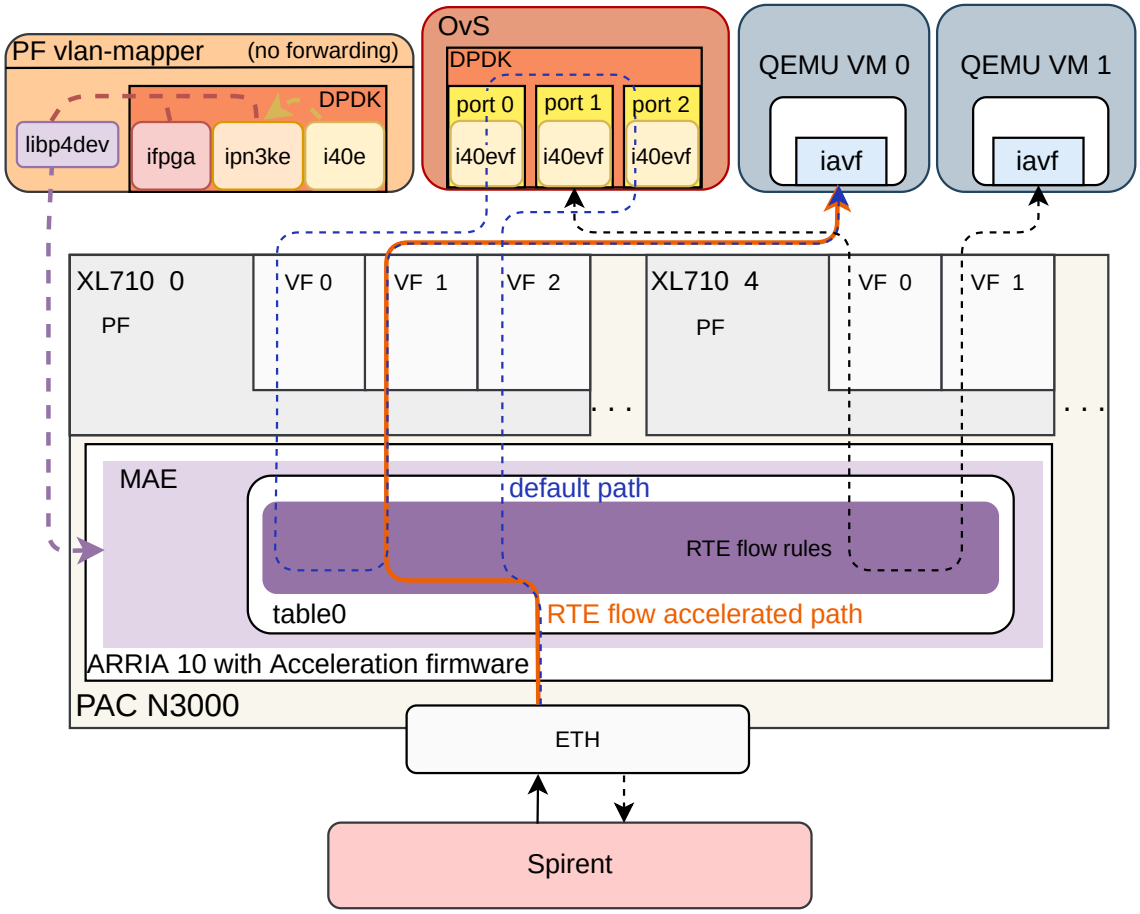
Měření PV spočívá v zasílání paketů z fyzického portu (ethernet) do virtuálního stroje, kde je měřena rychlost přibývajících paketů za jednotku času. V tomto případě bylo měření provedeno ve čtyřech bězích, ve kterých bylo OvS–DPDK nastaveno tak, jak je uvedeno v tabulce 3.6.

Číslo běhu	Počet jader přiřazených DPDK	Hardwarová akcelerace
1.	3	Vypnuta
2.	3	Zapnuta
3.	1	Vypnuta
4.	1	Zapnuta

Tabulka 3.6:

Jelikož OvS disponuje v tomto měření třemi porty, jeho propustnost bude optimální, když do DPDK budou do předány tři jádra nebo více. Běhy, ve kterých bylo DPDK přiděleno pouze jedno jádro (3. a 4.), byly provedeny pro zjištění schopnosti prototypu ušetřit spotřebu CPU.

Na obrázku 3.17 je vidět schéma měření v rámci topologie PV s hardwarově akcelerovaným OvS. RTE flow pravidla zde odkloňují pakety přijaté z ethernetového portu přímo do virtuálního stroje.

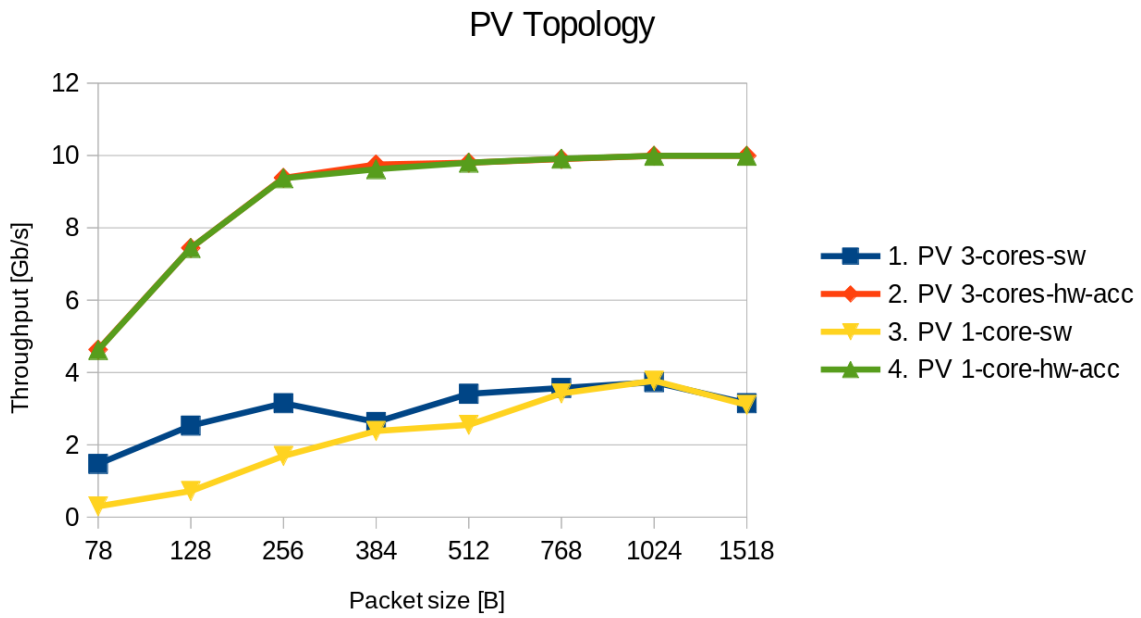


Obrázek 3.17: Schéma PV topologie u měření

Modře je opět naznačena cesta, kterou by byly pakety přenášeny bez použití hardwarové akcelerace. V tomto případě by byly přeposlány do OvS, odkud by směřovaly přes kartu do virtuálního stroje. Takže oproti akcelerované verzi HW → VM, je cesta v tomto případě prodloužena na HW → OvS → HW → VM. Tato cesta byla použita v běžích, kdy v OvS byla vypnuta hardwarová akcelerace (1. a 3. běh).

V grafu na obrázku 3.18 je možné vidět propustnosti u jednotlivých běhů OvS. Zde mají hardwarově akcelerované varianty větší propustnost, nicméně u menších paketů jde vidět menší propustnost, než je naměřena u PP testování. To je způsobeno přijímáním paketů ve virtuálním stroji, což je hlavní úzké hrdlo těchto variant.

Čistě softwarové varianty jsou zpomaleny softwarovém přepínání paketů. Zde se také projevuje, na rozdíl od hardwarově akcelerovaných variant, rozdílné množství jader přiřazených DPDK. Tento rozdíl je vidět zejména u menších paketů, kde se propustnost u jedno–



Obrázek 3.18: Výsledky měření u PV topologie

jádrové varianty snížila až dvojnásobně. To značí, že akcelerační prototyp je nejen schopen zvýšit propustnost, ale také dokáže efektivně fungovat bez vytížení tolika jader.

Zde je ještě nutné zmínit, že čistě softwarové varianty OvS v praxi nepoužívají pro připojení virtuálních strojů principu SR–IOV. Místo toho je použit framework VIRTIO, pomocí kterého je možné softwarově emulovat síťovou kartu a přiřadit ji virtuálnímu stroji. Tato softwarová varianta je výkonnější z hlediska propustnosti než používání SR–IOV bez hardwarového offloadu, nicméně o to více spotřebovává výkonu CPU.

Dá se tedy předpokládat, že VIRTIO varianta by s dostatečným množstvím CPU (3 a více) byla rychlejší než použití SR–IOV bez akcelerace. Nicméně i tak by byla omezena úzkým hrdlem, kterým je přijímání a přepínání paketů v software.

Kapitola 4

Závěr

V rámci této práce jsem popsal technologie potřebné pro akceleraci OvS. Na jejich základě jsem navrhl, implementoval a otestoval akcelerační prototyp využívající kartu PAC N3000 od společnosti Intel. Tento prototyp je založen na standartu SR–IOV a klasifikačním rozhraní RTE flow, které je součástí DPDK frameworku. Díky RTE flow je Open vSwitch schopen přenést klasifikační pravidla na úroveň karty PAC N3000. Testování prokázalo, že je tento prototyp schopen přepínat pakety běžného síťového provozu. Během měření se ukázalo, že nahraná RTE flow pravidla v kartě jsou schopna zvýšit propustnost OvS a je díky nim také možné snížit spotřebu procesoru.

Tento prototyp se odlišuje od klasických řešení akcelerace OvS zejména architekturou, kde jsou místo reprezentátorů používány virtuální funkce. To ve výsledku znamená použití dvojnásobného počtu virtuálních funkcí než v normálním případě. Tato odlišnost je způsobena chybějícím kódem v DPDK ovladači `i40e`, kvůli kterému nejsou schopny reprezentátory přenášet pakety. Pokud by byla tato vlastnost doimplementována ze strany společnosti Intel, byl by tento prototyp zjednodušen a fungoval by obvyklým způsobem. Nicméně i tak je schopen akcelerace OvS. Jediná limitující vlastnost je menší počet virtuálních strojů, které lze napojit na OvS, jelikož pro každý virtuální stroj jsou místo jedné virtuální funkce potřeba dvě. Toto omezení ale prozatím nepředstavuje velký problém, jelikož každé XL710 zařízení je schopné vytvořit až 32 virtuálních funkcí (po speciálním konfiguraci i 128). Pro kartu PAC N3000 s osmi XL710 zařízeními to znamená více než 100 virtuálních strojů, které je možné připojit pomocí karty k síti. To je při současné kapacitě serverů dostatečné.

Další věcí, ve které se tento prototyp liší od ostatních, je využití VFIO SR–IOV, které bylo přidáno do jádra operačního systému Linux v roce 2020. To znamená, že tento prototyp je jeden z prvních, který tuto technologii používá.

I když je tento prototyp schopen akcelerovat OvS, má i své meze. Ty spočívají zejména v akceleračním firmware, který je momentálně schopen poskytnout kapacitu v řádu stovek pravidel, které do něj OvS může nahrát. Rozšíření tohoto firmware o možnost nahrání řádově více pravidel (tisíce, desetitisíce) může být jedno z budoucích rozšířeních tohoto prototypu.

Literatura

[1] *Linux Drivers* [online]. [cit. 2022-01-02]. Dostupné z: https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html.

[2] *VFIO - "Virtual Function I/O"* [online]. [cit. 2022-01-01]. Dostupné z: https://www.kernel.org/doc/html/latest/driver-api/vfio.html.

[3] *Intel FPGA Programmable Acceleration Card N3000 Data Sheet*. 1.1. Intel, červen 2021.

[4] *Open vSwitch Documentation* [online]. 2021 [cit. 2021-21-12]. Dostupné z: http://docs.openvswitch.org.

[5] *QEMU Documentation* [online]. 2021 [cit. 2021-15-12]. Dostupné z: https://www.qemu.org/docs/master/index.html.

[6] *DPDK DOCUMENTATION* [online]. 2022 [cit. 2022-01-08]. Dostupné z: https://core.dpdk.org/doc/.

[7] *DPDK repositories* [online]. 2022 [cit. 2022-04-22]. Dostupné z: http://git.dpdk.org/dpdk/.

[8] *Intel® FPGA Programmable Acceleration Card N3000* [online]. 2022 [cit. 2022-01-19]. Dostupné z: https://www.intel.com/content/www/us/en/products/details/fpga/platforms/pac/n3000.html.

[9] DPDK. *RTE flow* [online]. [cit. 2022-01-03]. Dostupné z: https://doc.dpdk.org/guides/prog_guide/rte_flow.html.

[10] DPDK. *Environment Abstraction Layer* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html.

[11] DPDK. *I40E Poll Mode Driver* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://doc.dpdk.org/guides/nics/i40e.html.

[12] DPDK. *IPN3KE Poll Mode Driver* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://doc.dpdk.org/guides/nics/ipn3ke.html.

[13] DPDK. *Linux Drivers* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html.

[14] DPDK. *Network Interface Controller Drivers* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://doc.dpdk.org/guides/nics/index.html.

[15] DPDK. *Overview* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://doc.dpdk.org/guides/prog_guide/overview.html.

[16] DPDK. *Rawdev Drivers* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://doc.dpdk.org/guides/rawdevs/index.html.

[17] DPDK. *Supported Hardware* [online]. 2022 [cit. 2022-04-18]. Dostupné z: https://core.dpdk.org/supported/.

[18] DPDK. *Testpmd Runtime Functions* [online]. 2022 [cit. 2022-04-20]. Dostupné z: https://doc.dpdk.org/guides/testpmd_app_ug/testpmd_funcs.html.

[19] DVOŘÁK, M. *V2 - Modul pro přenos (offload) klasifikačních pravidel do akcelerační karty, Netcope Technologies, a.s.* 2020. Příloha k průběžné zprávě projektu TH04010193 za rok 2020.

[20] GORMAN, M. *Huge pages part 1 (Introdution)* [online]. 2010 [cit. 2022-01-19]. Dostupné z: https://lwn.net/Articles/374424/.

[21] IBM. *Hypervisors* [online]. [cit. 2021-19-12]. Dostupné z: https://www.ibm.com/topics/hypervisors.

[22] IBM. *Using Open vSwitch* [online]. [cit. 2021-21-12]. Dostupné z: https://www.ibm.com/docs/en/linux-on-systems?topic=choices-using-open-vswitch.

[23] INTEL. *Intel® Arria® 10 GT 1150 FPGA* [online]. [cit. 2022-01-04]. Dostupné z: https://www.intel.com/content/www/us/en/products/sku/210376/intel-arria-10-gt-1150-fpga/specifications.html.

[24] INTEL. *Intel® FPGAs & SoC FPGAs* [online]. [cit. 2022-01-04]. Dostupné z: https://www.intel.com/content/www/us/en/products/details/fpga.html.

[25] INTEL. *Intel® Network Adapter Driver for PCIe* 40 Gigabit Ethernet Network Connections under Linux** [online]. [cit. 2022-01-05]. Dostupné z: https://www.intel.com/content/www/us/en/download/18026/intel-network-adapter-driver-for-pcie-40-gigabit-ethernet-network-connections-under-linux.html.

[26] INTEL. *Intel® Network Adapter Linux* Virtual Function Driver for Intel® Ethernet Controller 700 and E810 Series* [online]. [cit. 2022-01-05]. Dostupné z: https://www.intel.com/content/www/us/en/download/18159/intel-network-adapter-linux-virtual-function-driver-for-intel-ethernet-controller-700-and-e810-series.html.

[27] INTEL. *Intel® Acceleration Stack User Guide: Intel FPGA Programmable Acceleration Card N3000* [online]. 2021 [cit. 2022-01-05]. Dostupné z: https://www.intel.com/content/www/us/en/docs/programmable/683040/1-1/about-this-document.html.

[28] INTEL. *Intel® Ethernet Converged Network Adapter XL710 10/40 GbE (Intel® Ethernet CNA XL710 10/40 GbE)* [online]. 2021 [cit. 2022-01-05]. Dostupné z: https://www.intel.com/content/www/us/en/products/docs/network-io/ethernet/network-adapters/ethernet-xl710-brief.html.

[29] Lacerda Ruivo, T. P. P. de et al. *Efficient High-Performance Computing with, Infiniband Hardware Virtualization* [online]. 2014 [cit. 2022-01-02]. Dostupné z: http://datasys.cs.iit.edu/reports/2014_IIT_virtualization-fermicloud.pdf.

[30] Libvirt. *Virtio* [online]. [cit. 2022-01-01]. Dostupné z: https://wiki.libvirt.org/page/Virtio.

[31] Luebbers, E., Liu, S. a Chu, M. *Simplify Software Integration for FPGA Accelerators with OPAE* [online]. [cit. 2022-01-05]. Dostupné z: https://01.org/sites/default/files/downloads/opae/open-programmable-acceleration-engine-paper.pdf.

[32] McGillicuddy, S. *Virtual Switch* [online]. [cit. 2021-20-12]. Dostupné z: https://searchservervirtualization.techtarget.com/definition/virtual-switch.

[33] Microsoft. *What is a virtual machine (VM)* [online]. [cit. 2021-19-12]. Dostupné z: https://azure.microsoft.com/en-us/overview/what-is-a-virtual-machine/.

[34] NVIDIA. *OVS Offload Using ASAP2 Direct* [online]. [cit. 2022-01-03]. Dostupné z: https://docs.mellanox.com/display/MLNXENv493150/OVS+Offload+Using+ASAP2+Direct.

[35] Oracle. *Reasons to Use Virtualization* [online]. [cit. 2021-20-12]. Dostupné z: https://docs.oracle.com/cd/E27300_01/E27309/html/vmusg-virtualization-reasons.html.

[36] Orsák, M. a Beneš, T. High-speed stateful packet classifier based on TSS algorithm optimized for off-chip memories. In: *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*. 2021, s. 151–156. DOI: 10.1109/DDECS52668.2021.9417060.

[37] RedHat. *Guest virtual machine device configuration* [online]. [cit. 2022-01-02]. Dostupné z: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/virtualization_administration_guide/chap-guest_virtual_machine_device_configuration.

[38] RedHat. *Red Hat's perspective on OVS HW Offload Status* [online]. [cit. 2022-01-01]. Dostupné z: https://www.openvswitch.org/support/ovscon2017/khan.pdf.

[39] RedHat. *SRIOV* [online]. [cit. 2022-01-02]. Dostupné z: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/virtualization_host_configuration_and_guest_installation_guide/chap-virtualization_host_configuration_and_guest_installation_guide-sr_iov.

[40] RedHat. *Virtio* [online]. [cit. 2022-01-01]. Dostupné z: https://www.linux-kvm.org/page/Virtio.

[41] Redhat. *What is a virtual machine* [online]. [cit. 2021-15-12]. Dostupné z: https://www.redhat.com/en/topics/virtualization/what-is-a-virtual-machine.

[42] Redhat. *What is KVM* [online]. [cit. 2021-19-12]. Dostupné z: https://www.redhat.com/en/topics/virtualization/what-is-KVM.

[43] Smith, R. *Using QEMU for cross-platform development* [online]. [cit. 2021-20-12]. Dostupné z: https://developer.ibm.com/tutorials/l-qemu-development/.

[44] SUSE. *Running Virtual Machines with qemu-kvm* [online]. [cit. 2021-20-12]. Dostupné z: https://documentation.suse.com/sles/11-SP4/html/SLES-all/cha-qemu-running.html.

[45] VMWARE. *Add a Network Adapter to a Virtual Machine* [online]. [cit. 2021-20-12]. Dostupné z: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.vm_admin.doc/GUID-47765AC6-38AA-4ADB-84DF-5094EDA718EC.html.

[46] WILLIAMSON, A. *Vfio/pci: SR-IOV support* [online]. 2020 [cit. 2022-01-03]. Dostupné z: https://lore.kernel.org/lkml/158085337582.9445.17682266437583505502.stgit@gimli.home.