VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

AKCELERACE OVS S VYUŽITÍM AKCELERAČNÍ KARTY S FPGA

OVS ACCELERATION USING FPGA ACCELERATION CARD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MATEJ VIDO**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAN KOŘENEK, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Vido Matej, Bc.**
Obor: Bezpečnost informačních technologií
Téma: **Akcelerace OVS s využitím akcelerační karty s FPGA**
OVS Acceleration Using FPGA Acceleration Card
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se s technologií OVS a nízkoprofilovou akcelerační kartou COMBO 100G.
2. Nastudujte způsob akcelerace OVS pomocí systému DPDK.
3. Navrhněte vhodný způsob akcelerace OVS s využitím akcelerační karty COMBO 100G.
4. Proveďte implementaci navrženého způsobu akcelerace.
5. Ověřte parametry vytvořené implementace.
6. Diskutujte dosažené výsledky a možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

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

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

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

Vedoucí: **Kořenek Jan, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

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

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Výkon virtuálneho prepínača Open vSwitch (OVS) je nedostatočný pre súčasné požiadavky na kapacitu liniek pre pripojenie serverov. Preto vznikajú snahy o akceleráciu OVS či už na úrovni softvéru, ale aj prenesením záťaže do sieťovej karty. V tejto práci je riešená problematika akcelerácie OVS s využitím sieťových kariet COMBO pre 100G Ethernet vyvíjaných združením CESNET. Navrhnuté riešenie využíva firmvér pre FPGA generovaný z popisu v jazyku P4 pre klasifikáciu paketov v karte a framework DPDK pre dátové prenosy a nahrávanie klasifikačných pravidiel do karty. Pri preposielaní jedného dátového toku z fyzického rozhrania na fyzické rozhranie s využitím jedného jadra CPU je pri najkratších rámcoch dosahovaná rýchlosť prenosu paketov 11,2 Mp/s (10-krát viac ako s klasickým OVS) s klasifikáciou v karte a 5,9 Mp/s bez klasifikácie v karte.

Abstract

The performance of the virtual switch Open vSwitch (OVS) is insufficient to satisfy the current requirements for link bandwidth of the server connections. There is an effort to accelerate the OVS both in the software and in the hardware by offloading the datapath to the smart network interface cards. In this work the COMBO card for 100G Ethernet developed by CESNET is used to accelerate the OVS. The suggested solution utilizes the firmware for FPGA generated from the definition in the P4 language to classify the packets in the card and DPDK for the data transfers and offloading the classification rules into the card. Forwarding of one flow with the shortest frames from physical to physical interface using one CPU core reaches forwarding rate of 11.2 Mp/s (10 times more than the standard OVS) with classification in the card and 5.9 Mp/s without classification in the card.

Kľúčové slová

Open vSwitch, OVS, DPDK, rte_flow, P4, FPGA, akcelerácia, COMBO, 100G Ethernet

Keywords

Open vSwitch, OVS, DPDK, rte_flow, P4, FPGA, acceleration, COMBO, 100G Ethernet

Citácia

VIDO, Matej. *Akcelerace OVS s využitím akcelerační karty s FPGA*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jan Kořenek, Ph.D.

Akcelerace OVS s využitím akcelerační karty s FPGA

Prehlásenie

Vyhlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pána Ing. Jana Kořenka, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Matej Vido
21. mája 2018

Poďakovanie

Rád by som poďakoval vedúcemu Ing. Janovi Kořenkovi, Ph.D. za venovaný čas a odbornú pomoc pri vypracovaní práce. Ďalej by som chcel poďakovať členom tímu z projektu Liberouter za príjemné a motivačné pracovné prostredie. V neposlednom rade ďakujem rodine a priateľom za podporu pri štúdiu.

Obsah

Kapitola 1

Úvod

Tradičné architektúry sietí zložených z vrstiev ethernetových smerovačov a prepínačov nevyhovujú súčasným dynamickým požiadavkám zo strán veľkých dátových centier. Problémom týchto hierarchických architektúr je prílišná zložitosť spôsobená množstvom sieťových protokolov a zariadení, ktoré treba brať do úvahy pri zmenách v sieti. Kvôli zložitosti súčasných sietí je náročné udržiavať konzistentné prístupové, bezpečnostné a rôzne iné politiky vo veľkých sieťach, čo môže mať za následok vystavenie sietí bezpečnostným rizikám a iným negatívnym dôsledkom. S neustále rastúcimi požiadavkami na dátové centrá musia držať krok aj siete. Komplexné statické siete majú problémy so škálovateľnosťou.

Riešením na problémy klasických hierarchických sietí je koncept softvérovo definovaných sietí[1] (SDN), ktorý bol predstavený neziskovou organizáciou *Open Networking Foundation* (ONF) [27]. ONF podporovaná významnými poskytovateľmi internetových služieb a telekomunikačnými spoločnosťami riadi presadzovanie SDN a štandardizáciu dôležitých častí architektúry SDN.

Na rozdiel od klasických sieťových zariadení, v ktorých je kontrolná a dátová vrstva súčasťou jedného zariadenia, v architektúre SDN sú kontrolná a dátová vrstva oddelené. Pre aplikácie a sieťové služby je infraštruktúra siete abstrahovaná a sieť môže byť vnímaná ako logický celok. Celkový pohľad na sieť je udržovaný v SDN radiči, ktorý aplikáciám poskytuje jednotné rozhranie pre riadenie činnosti siete nezávisle na sieťových zariadeniach, z ktorých sa sieť skladá. Toto zjednodušuje zároveň aj nároky na sieťové zariadenia, ktorým stačí prijímať inštrukcie od SDN radiča, podľa ktorých prepínajú dátovú prevádzku.

S konceptom SDN úzko súvisí pojem virtualizácia sieťových funkcií[2] (NFV) predstavený štandardizačnou organizáciou ETSI[3] [10]. NFV sa vhodne dopĺňa s konceptom SDN avšak obe môžu byť použité aj nezávisle na druhom. Myšlienka NFV je využitie bežných virtualizačných techník k softvérovej implementácii rôznych sieťových zariadení, ktoré môžu bežať na štandardných serveroch. Virtualizované môžu byť všetky typy sieťových zariadení. Takéto virtualizované zariadenie sa potom označuje ako virtuálna sieťová funkcia[4] (VNF). Ako príklad je možné uviesť: smerovač, firewall, vyrovnávač záťaže (*Load Balancer*), preklad sieťových adries (*Network Address Translation* – NAT), kvalita služieb (*Quality of Service* – QoS), hlboká kontrola paketov (*Deep Packet Inspection* – DPI). Han et al. zanalyzovali výzvy, ktoré prínaša použitie NFV z hľadiska výkonu, ovládateľnosti, spoľahlivosti, stabi-

[1] *Software-Defined Networking*
[2] *Network Functions Virtualization*
[3] *European Telecommunications Standards Institute*
[4] *Virtual Network Function*

lity a bezpečnosti [15]. Wang et al. zostrojili experimentálnu NFV platformu z bežných technológií pre NFV, na ktorej preskúmali a vyhodnotili výkonnosť a škálovateľnosť [41].

VNF môže bežať na jednom alebo viacerých virtuálnych strojoch. Pre správu behu virtuálnych strojov sa používa softvér nazývaný hypervízor, ktorý zabezpečuje abstrakciu hardvérových prostriedkov pre virtuálne stroje. Každá VNF spracováva určitým spôsobom dáta zo siete. Prístup k dátam zo siete môže hypervízor sprostredkovávať viacerými spôsobmi, ktoré popisujú Rosenblum a Waldspurger [35]. Hypervízor potrebuje k svojej činnosti virtuálny prepínač, ktorý riadi prepínanie paketov medzi virtuálnymi strojmi a fyzickým hardvérom.

Dostupné sú viaceré riešenia virtuálnych prepínačov, ktoré sa líšia svojou výkonnosťou a ponúkanou funkcionalitou. Medzi široko používané riešenia s otvorenými zdrojovými kódmi patrí *Open vSwitch* (OVS) [48]. Rozšírenými proprietárnymi alternatívami sú *Nexus 1000V* [11, 12] od Cisca a *vNetwork Distributed Switch* [40] od VMware. Výkonnosť OVS bola analyzovaná a porovnávaná s inými riešeniami vo viacerých prácach. Sans a Gamess porovnávali výkon klasického prepínača Cisco Catalyst 3750 s virtuálnymi prepínačmi OVS a LiSA [36]. Emmerich et al. zanalyzovali výkon OVS v rôznych prípadoch použitia [14, 13]. Paolino et al. porovnali výkonnosť OVS s virtuálnym prepínačom VOSYSwitch [32].

Kvôli rastúcim požiadavkám na priepustnosť sú vyvíjané snahy o akceleráciu OVS. K akcelerácii sa používa framework pre rýchle spracovanie paketov v užívateľskom priestore operačného systému – Data Plane Development Kit (DPDK) [42], s ktorým je možné dosahovať vyššiu priepustnosť a nižšiu latenciu ako v klasickom OVS, kedy je dátová cesta spracovávaná modulom jadra. Avšak aj s použitím DPDK je na spracovanie dátovej prevádzky potrebné vyčleniť niekoľko jadier CPU, čím sa znižuje efektivita využitia hardvérových prostriedkov pre VNF. V technickej správe [17] od spoločnosti Intel je dostupné porovnanie výkonnosti OVS a riešenia využívajúceho DPDK. Viacerí výrobcovia inteligentných sieťových kariet, medzi ktorých patria napríklad Cavium [7], Mellanox Technologies [20] a Netronome [24, 16], predstavili rôzne riešenia prenesenia spracovania paketov (*offload*) do sieťových kariet, čím je možné dosahovať zníženie zaťaženia procesoru a zvýšenie priepustnosti OVS.

Združenie CESNET vyvíja akceleračné karty COMBO s FPGA čipmi a rozhraniami pre Ethernet až do rýchlosti 100 Gb/s [8]. Tieto karty sú určené pre akceleráciu sieťových operácií ako napríklad monitorovanie a filtrácia sieťových tokov. Karty COMBO umožňujú rýchle DMA prenosy dát do operačnej pamäte a spracovanie softvérom pri plnej rýchlosti linky aj pre najkratšie ethernetové rámce.

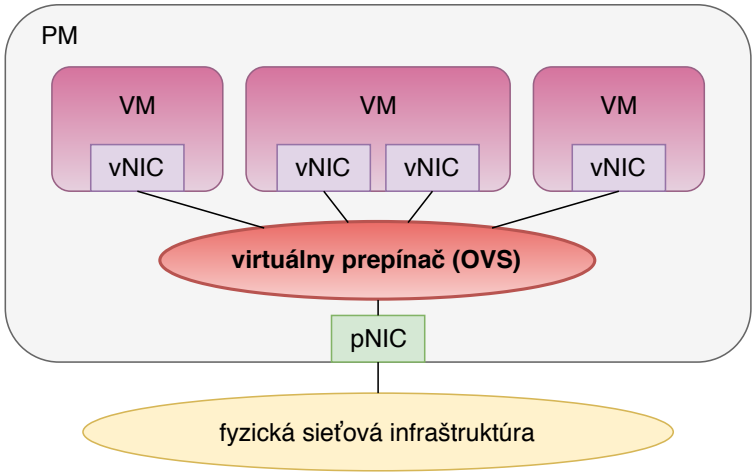
Cieľom tejto práce je analýza súčasných prístupov k akcelerácii OVS a možností akceleračných kariet COMBO, na základe ktorej je navrhnuté a implementované riešenie akcelerácie OVS s využitím kariet COMBO. Výkonnosť implementovaného riešenia je odmeraná a vyhodnotená pre rôzne konfigurácie v dvoch topológiách zapojenia: fyzické rozhranie – fyzické rozhranie, fyzické rozhranie – virtuálny stroj – fyzické rozhranie.

V kapitole 2 je popísaná architektúra, princíp činnosti OVS a metodika testovania virtuálnych prepínačov. Akcelerácia OVS s využitím frameworku DPDK a princípy akcelerácie prenesením záťaže do sieťovej karty sú popísané v kapitole 3. Kapitola 4 popisuje akceleračné karty COMBO a jazyk P4 pre popis sieťového zariadenia, z ktorého je možné generovať firmvér pre karty COMBO. V kapitole 5 je popísaný spôsob podpory kariet COMBO v DPDK a zhodnotenie výkonnosti dátových prenosov s kartami COMBO cez DPDK. Návrh a popis riešenia akcelerácie OVS so sieťovými kartami COMBO je popísaný v kapitole 6. Merania výkonnosti a výsledky sú popísané v kapitole 7. Zhrnutie a možnosti pokračovania práce sú v kapitole 8.

Kapitola 2

Open vSwitch

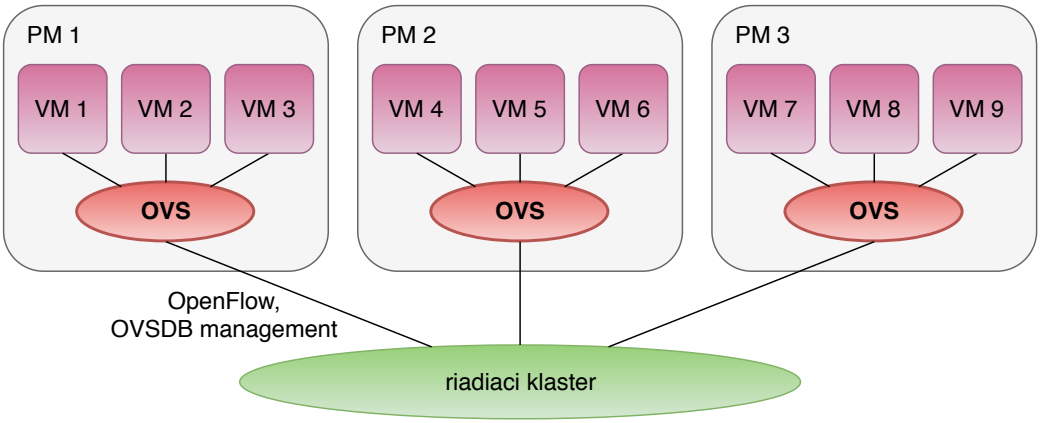
Open vSwitch (OVS) je viacvrstvový virtuálny prepínač licencovaný pod open source licenciou Apache 2.0, ktorý podporuje štandardné spravovacie rozhrania a protokoly [48]. OVS je používaný v mnohých produktoch a produkčných prostrediach a bol prispôsobený na viaceré virtualizačné platformy – XenServer 6.0, Xen Cloud Platform, Xen, KVM, Proxmox VE a VirtualBox. OVS je taktiež integrovaný do viacerých systémov pre správu virtualizovaných prostredí vrátane OpenStack, openQRM, OpenNebula a oVirt. Medzi podporované operačné systémy patria okrem Linuxu aj FreeBSD, NetBSD, Citrix XenServer a Windows.



Obr. 2.1: Schéma použitia virtuálneho prepínača vo virtualizovanom prostredí.

Vo virtualizovaných prostrediach je potrebné zabezpečiť prepojenie virtuálnych strojov bežiacich na rovnakom fyzickom stroji medzi sebou a zároveň s fyzickou sieťou. Súčasťou hypervízora je softvérový prepínač, ktorý takéto prepojenie zabezpečuje. OVS je jeden z dostupných virtuálnych prepínačov, ktoré sa v hypervízoroch na prepínanie paketov používajú. Na obrázku 2.1 je schéma, ktorá zobrazuje typické použitie virtuálneho prepínača. Softvérový virtuálny prepínač prepája virtuálne sieťové karty (vNIC) virtuálnych strojov (VM) s fyzickými sieťovými kartami (pNIC) fyzického stroja (PM), na ktorom virtuálne stroje bežia, a tak sú virtuálne stroje pripojené do fyzickej siete. V typických aplikáciách vo virtualizovaných prostrediach sú pakety prepínané z *pNIC* do *vNIC*, z *vNIC* do *pNIC* a z *vNIC* do *vNIC* [13].

V hypervízoroch založených na Linuxe je možné na prepínanie paketov medzi virtuálnymi strojmi a fyzickým rozhraním použiť L2 prepínač vstavaný v jadre (*Linux Bridge*) [53].



Obr. 2.2: Schéma distribuovaného nasadenia OVS v prostredí s viacerými fyzickými strojmi.

Avšak v nasadeniach, kde je na virtualizáciu použitých viac serverov, nie je vstavaný *Linux Bridge* vhodný. OVS je zameraný na použitie v distribuovaných prostrediach s veľkým počtom serverov (PM), ktoré sú charakteristické veľmi dynamickou architektúrou a potrebou udržiavať logické abstrakcie. Na obrázku 2.2 je znázornené použitie OVS v distribuovanom prostredí, kedy sú jednotlivé inštancie OVS na rôznych fyzických strojoch riadené z riadiaceho klastru pomocou protokolov OpenFlow a protokolu pre správu OVSDB databázy. Pre úspešnú adaptáciu OVS v takýchto prostrediach sú dôležité najmä nasledujúce vlastnosti.

Stavová mobilita

Kompletný sieťový stav asociovaný so sieťovou entitou (zahŕňajúci napríklad záznam v L2 tabuľke, preposielací L3 stav, smerovací stav, zoznamy kontroly prístupu, politiky kvality služieb, monitorovacia konfigurácia) by mal byť identifikovateľný a migrovateľný medzi rôznymi fyzickými hostiteľskými strojmi. OVS podporuje konfiguráciu a migráciu pomalého (konfigurácia zariadenia) aj rýchleho sieťového stavu medzi jednotlivými inštanciami.

Reagovanie na dynamiku siete

Vo virtualizovaných prostrediach je typické veľké množstvo rýchlych zmien – vytváranie a rušenie virtuálnych strojov, posúvanie virtuálnych strojov v čase dopredu a dozadu, zmeny v prostredí logických sietí. OVS podporuje radu funkcií, ktoré umožňujú systému reagovať a prispôsobovať sa zmenám v sieti. Okrem základnej podpory účtovateľnosti a sledovania siete prostredníctvom protokolov NetFlow, IPFIX a sFlow obsahuje OVS databázu sieťového stavu, ktorá podporuje vzdialené spúšťače. Orchestračný softvér tak môže sledovať sieť z viacerých hľadísk a reagovať na ich zmeny.

Udržiavanie logických značiek

Distribuované virtuálne prepínače udržiavajú logický kontext v sieti pomocou pripájania značiek k paketom. Značky slúžia na jednoznačnú identifikáciu virtuálneho stroja alebo udržiavanie iného kontextu dôležitého vrámci logickej domény. OVS obsahuje viaceré metódy pre špecifikáciu a udržiavanie značkovacích pravidiel, ktoré sú dostupné vzdialeným orchestračným procesom. Ukladanie značkovacích pravidiel je optimalizované tak, aby bolo možné konfigurovať, meniť a migrovať tisícky značkovacích a mapovacích pravidiel.

Integrácia s hardvérom

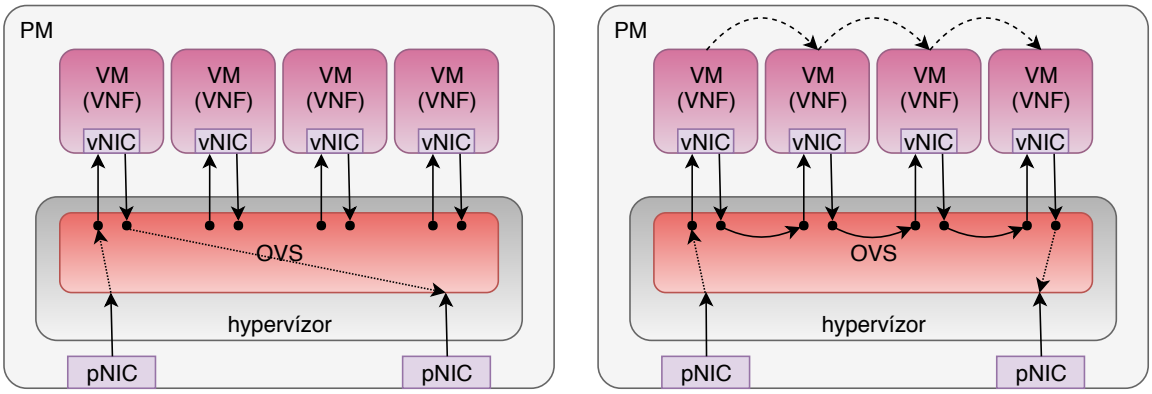
Dátová cesta v OVS je navrhnutá tak, aby bolo možné využiť hardvérové prostriedky na prepínanie paketov, či už sa jedná o fyzické prepínače alebo sieťové karty. Výhodou hardvérovej integrácie je možnosť dosiahnutia vyššej výkonnosti vo virtuálnych prostrediach.

2.1 Charakter sieťovej prevádzky

Podľa smeru toku dát vo virtualizovanom prostredí sa dá charakter sieťovej prevádzky rozdeliť na dva typy: sever-juh (*north-south*) a východ-západ (*east-west*) [18]. Tieto dva typy charakteru sieťovej prevádzky sú zobrazené na obrázku 2.3.

Dátová prevádzka typu sever-juh je charakteristická tým, že dáta prichádzajú do fyzického stroja (PM) zvonku, sú nasmerované do virtuálneho stroja (VM), kde ich spracuje virtuálna sieťová funkcia (VNF), a potom sú opäť poslané von do siete cez fyzické rozhranie. Príkladom virtuálnej sieťovej funkcie, ktorá má takýto charakter sieťovej prevádzky je smerovač.

Pri sieťovej prevádzke typu východ-západ sú dáta spracované jednou virtuálnou funkciou vo virtuálnom stroji, potom ďalšou, ktorá beží na tom istom fyzickom stroji a takto sa to môže opakovať niekoľkokrát, pričom viaceré virtuálne funkcie sa nachádzajú na tom istom fyzickom stroji. Príkladom je reťazenie sieťových služieb.



(a) sever-juh (*north-south*) (b) východ-západ (*east-west*)

Obr. 2.3: Schémy charakteru sieťovej prevádzky podľa smeru toku dát.

2.2 Architektúra OVS

Kľúčovými požiadavkami pri návrhu OVS bola výkonnosť požadovaná produkčnými prostrediami, v ktorých sa OVS nasadzuje, a programovateľnosť potrebná pri virtualizácii sietí. V bežných sieťových zariadeniach je možné vysokú výkonnosť dosiahnuť špecializáciou či už v hardvéri alebo proprietárnym softvérom bežiacim na zariadeniach daného výrobcu. OVS musel byť navrhnutý tak, aby podporoval flexibilitu a všeobecné použitie na rôznych platformách, a zároveň umožňoval dosahovať vysokú výkonnosť.

Pracovné prostredia virtuálnych prepínačov a klasických sieťových zariadení sa líšia. Z ich odlišností vychádzali obmedzenia a výzvy, ktoré ovyplyvnili návrh architektúry OVS [34]. V klasických sieťových zariadeniach je uprednostňované vyhradenie hardvérových

zdrojov na dosiahnutie výkonnosti umožňujúcej zvládať rýchlosť linky v najhoršom prípade. Virtuálny prepínač však potrebuje využívať zdroje efektívne, aby zostali dostatočné prostriedky pre hlavnú úlohu hypervízora – beh funkcií vo virtuálnom stroji. Sieťovanie vo virtuálnom prostredí je teda narozdiel od fyzického prostredia optimalizované na bežné prípady použitia. Pri návrhu OVS toto viedlo k použitiu rôznych vyrovnávacích pamätí, ktoré umožňujú dosiahnuť menšie zaťaženie procesoru v bežných podmienkach.

Umiestnenie virtuálnych prepínačov na okraji sietí viedlo ku komplikácii škálovateľnosti. Pre virtuálne prepínače je bežné, že obsahujú tisícky *point-to-point* IP tunelov s ďalšími virtuálnymi prepínačmi ostatných hypervízorov. Virtuálne prepínače dostávajú aktualizácie preposielacích stavov, keď sa zapínajú, migrujú a vypínajú virtuálne stroje. Zmeny v ostatných hypervízoroch môžu mať vplyv na lokálny stav virtuálneho prepínača, ktorý sa môže neustále meniť. Toto malo vplyv na návrh klasifikačného algoritmu, ktorý bol zostrojený tak, aby aktualizácie mali konštantnú zložitosť.

Niektoré virtuálne prepínače majú pevne určenú zreťazenú linku spracovania paketov. Nastavovať sa dajú iba pevne určené vlastnosti. Narozdiel od toho OVS bol od počiatku navrhnutý ako OpenFlow prepínač, vďaka čomu je preprogramovateľný prostredníctvom protokolu OpenFlow. V pokročilejších prípadoch použitia však toto vedie na dlhé zreťazené linky spracovania paketov a vo výsledku väčšie zaťaženie pri klasifikácii, kvôli čomu bolo nutné v OVS implementovať použitie vyrovnávacích pamätí pre toky.

2.2.1 Štruktúra

Na obrázku 2.4 je znázornená štruktúra OVS. Prepínanie paketov v OVS zabezpečujú dve najdôležitejšie komponenty `ovs-vswitchd` – démon v užívateľskom priestore, ktorý sa nelíši pre rôzne operačné systémy, a modul pre dátovú cestu v jadre operačného systému, ktorý je špecifický pre rôzne operačné systémy. Ďalšou dôležitou súčasťou OVS je `ovsdb-server` [49], čo je program, ktorý implementuje protokol pre správu OVS databázy[1] [33] a poskytuje RPC[2] rozhranie pre OVS databázy. Podporuje pripojenie JSON-RPC klientov cez aktívne alebo pasívne TCP/IP alebo unixové schránky.

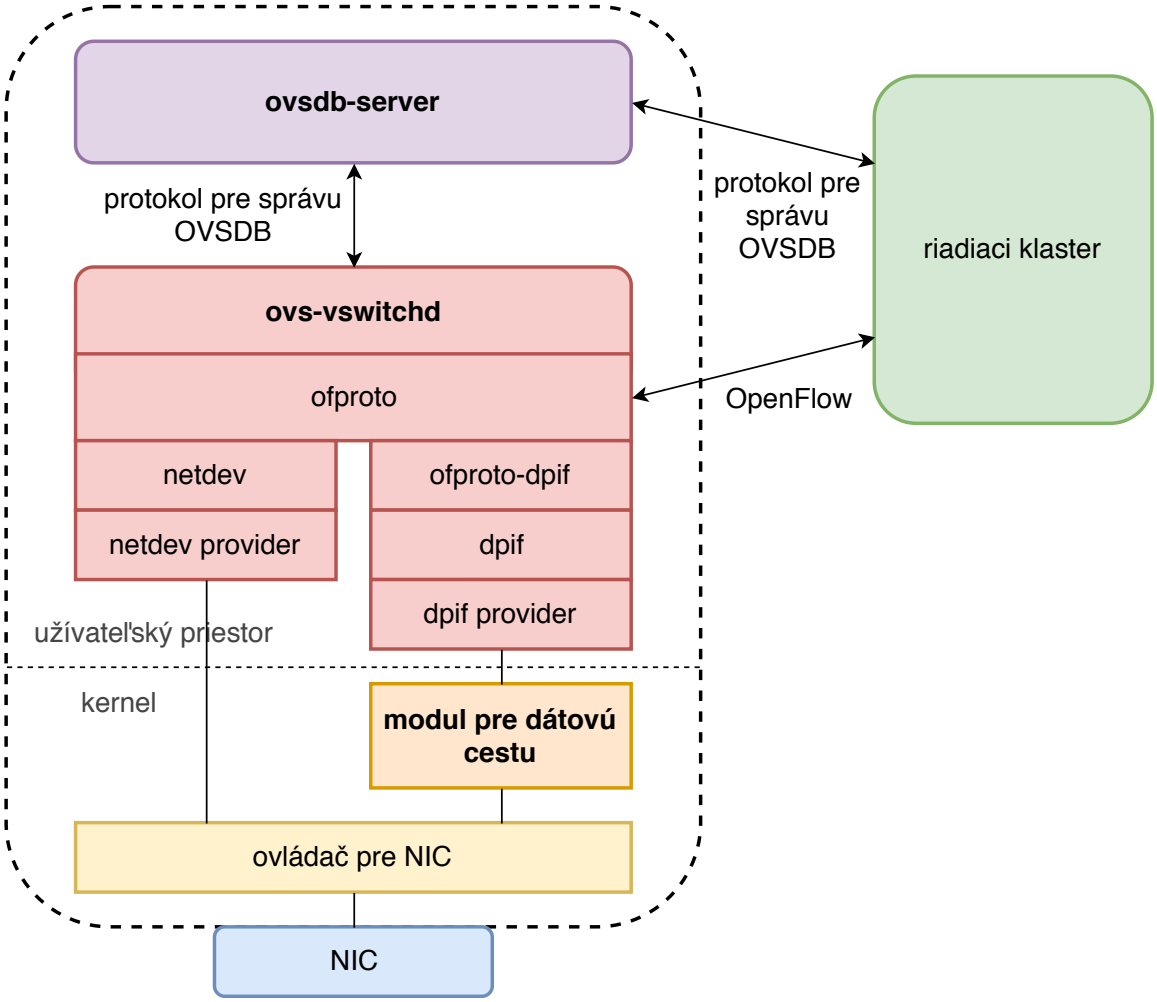
`ovs-vswitchd` demón sa zo štrukturálneho hľadiska dá rozdeliť na viac častí, ktoré sú dôležité pri pridávaní podpory nového hardvéru alebo softvéru [51]. `ovs-vswitchd` načítava požadovanú konfiguráciu OVS z `ovsdb-server`u a predáva ju ďalej knižnici `ofproto`, z ktorej zase posúva určité stavové a štatistické informácie do databázy.

Knižnica `netdev` abstrahuje interakciu so sieťovými zariadeniami. Rozhranie pre rôzne prístupy k sieťovým zariadeniam poskytuje `netdev provider`. Pre každý port v prepínači potrebuje OVS vytvoriť `netdev` štruktúru, k čomu je potrebné, aby bol pre daný hardvér prípadne softvér implementovaný `netdev provider`.

Knižnica `ofproto` implementuje OpenFlow prepínač. Cez sieť komunikuje s OpenFlow radičom a rozhranie k rôznym hardvérovým alebo softvérovým implementáciám prepínača poskytuje `ofproto provider`.

OVS obsahuje vstavaný `ofproto provider`, ktorý sa nazýva `ofproto-dpif`. Tento je postavený na knižnici `dpif`, ktorá komunikuje s dátovou cestou. Rozhranie k dátovej ceste poskytuje `dpif provider`. V Linuxe je to `dpif-netlink` na komunikáciu s implementáciou dátovej cesty v module jadra, pri ktorej sa používajú *netlink* schránky, a `dpif-netdev` pre dátovú cestu implementovanú v užívateľskom priestore, ktorá prijíma pakety cez `netdev`.

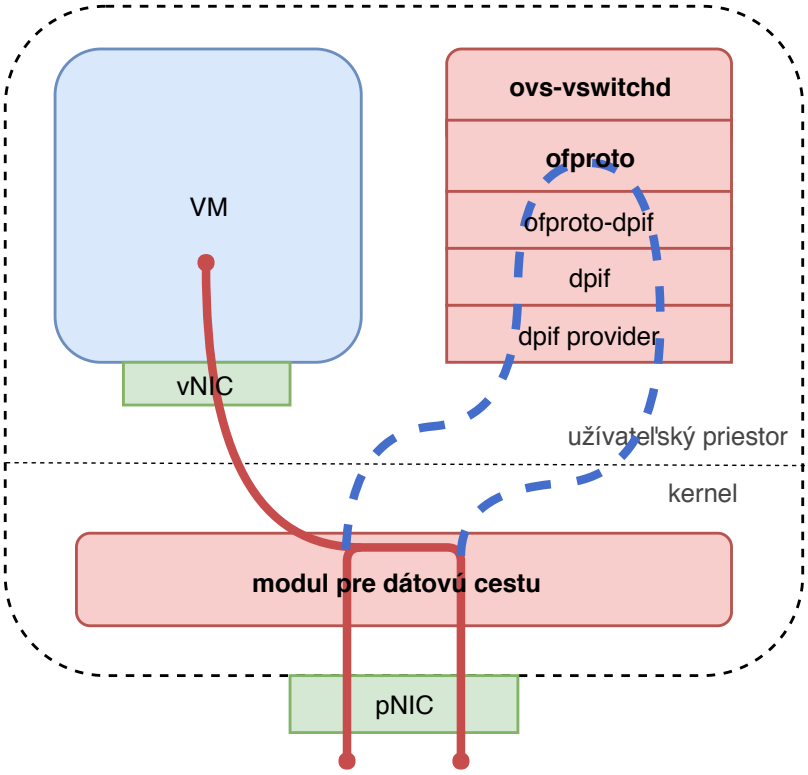
[1] *Open vSwitch Database Management Protocol*
[2] *Remote Procedure Call*



Obr. 2.4: Schéma komponent, z ktorých sa skladá Open vSwitch.

Dôležitou súčasťou OVS sú aj nástroje umožňujúce ovládanie a konfiguráciu užívateľského démona, databázy a dátovej cesty, medzi ktoré patria napríklad:

- `ovsdb-client` – klient poskytujúci rozhranie v príkazovom riadku pre komunikáciu s bežiacim `ovsdb-server`om
- `ovsdb-tool` – nástroj v príkazovom riadku pre správu súborov s OVS databázou, ktorá sa používa napríklad na vytvorenie a inicializáciu súboru s databázou
- `ovs-appctl` – nástroj na zasielanie príkazov bežiacim OVS démonom, ktoré umožňujú napríklad kontrolu ich funkčnosti alebo získanie ich nastavení
- `ovs-ctl` – pomocný skript pre spúšťanie OVS démona a `ovsdb-server`u
- `ovs-dpctl` – nástroj na vytváranie, zmenu a rušenie dátových ciest používaný na správu dátových ciest mimo užívateľského démona, t. j. v jadre operačného systému
- `ovs-ofctl` – nástroj pre monitorovanie a administráciu OpenFlow prepínačov, ktorý umožňuje aj zobrazovanie aktuálneho stavu OpenFlow prepínača
- `ovs-pki` – nástroj pre vytváranie a spravovanie infraštruktúry verejných kľúčov
- `ovs-vsctl` – nástroj na konfiguráciu `ovs-vswitchd` démona



Obr. 2.5: Priechod paketu dátovou cestou OVS. Červená čiara znázorňuje rýchlu cestu. Modrá čiarkovaná čiara znázorňuje pomalú cestu, kedy musel byť paket poslaný do užívateľského démona.

V referenčnej príručke [52] je možné nájsť podrobný zoznam vrátane manuálov k nástrojom, ktoré distribúcia OVS obsahuje.

2.2.2 Dátová cesta

Obrázok 2.5 zobrazuje postup spracovania paketov v OVS. V príklade na obrázku sa jedná o preposlanie paketu z fyzického rozhrania (pNIC) na virtuálne rozhranie (vNIC) a naopak.

Modul jadra pre dátovú cestu ako prvý obdrží pakety prichádzajúce z fyzickej sieťovej karty alebo z virtuálneho stroja. Modul pre dátovú cestu obsahuje vyrovnávacie pamäte pre toky. Ak sa v nich nachádza informácia, čo spraviť s paketom daného typu, modul vykoná príslušné akcie. Akcie môžu byť viacerých typov: preposlanie paketu na určité porty alebo tunely, upravenie hlavičiek paketu, uloženie štatistík, zahodenie paketu. Jedná sa o rýchlu cestu[3], ktorá je na obrázku 2.5 vyznačená červenou čiarou.

V prípade, že sa vo vyrovnávacích pamätiach modulu nenájde zhoda pre daný paket, modul pošle paket do `ovs-vswitchd`, ktorý podľa svojich tabuliek tokov rozhodne, ako s daným paketom naložiť. Paket je potom poslaný späť do modulu pre dátovú cestu vrátane informácií, čo s daným paketom spraviť, ktoré modul vykoná. Tento postup sa nazýva pomalá cesta[4]. Na obrázku 2.5 je znázornená modrou prerušovanou čiarou. `ovs-vswitchd` taktiež inštruuje modul, aby si uložil akcie pre daný typ paketu do svojich vyrovnávacích

[3] *fast path*
[4] *slow path*

pamätí. Ďalšie pakety rovnakého toku sú potom už nájdené vo vyrovnávacej pamäti modulu pre dátovú cestu.

2.2.3 Vyrovnávacie pamäte

Mechanizmus vyrovnávacích pamätí pre toky v OVS prešiel v priebehu času výrazným vývojom. Na začiatku vývoja OVS v roku 2007, bolo možné dosiahnuť dobrú výkonnosť prepínania paketov iba v jadre [34]. Preto bolo celé spracovanie OpenFlow implementované v module jadra. Paket bol prijatý zo sieťovej karty alebo virtuálneho stroja, klasifikovaný pomocou OpenFlow tabuľky a poslaný na ďalší port. Tento prístup sa ukázal ako nevhodný kvôli náročnosti vývoja v jadre, distribúcii a aktualizácii modulov jadra. Taktiež nebolo možné, aby sa modul s implementáciou OpenFlow stal súčasťou hlavnej vývojovej vetvy[5] Linuxu.

Microflow cache Riešením bola implementácia vyrovnávacej pamäte pre toky, v ktorej sa hľadala presná zhoda záznamu v tabuľke so všetkými položkami hlavičiek paketu podporovanými protokolom OpenFlow. Táto vyrovnávacia pamäť sa nazýva *microflow cache*.

Microflow cache umožnila zásadné zjednodušenie, pretože v module jadra stačilo implementovať jednoduchú hešovaciu tabuľku namiesto zložitého klasifikátoru podporujúceho ľubovoľné položky a maskovanie. Ak však nenastala presná zhoda na všetky položky záznamu v tabuľke, paket bol presunutý do užívateľského priestoru, kde sa v skutočnej OpenFlow tabuľke vyhľadali záznamy pre daný paket a rozhodlo sa, čo sa s daným paketom vykoná.

Dôležitým ukazovateľom výkonu bola teda doba[6] potrebná na náhlasenie výpadku[7] v *microflow cache* a získanie reakcie z užívateľského priestoru. Na zníženie tejto doby bolo aplikovaných niekoľko techník ako dávkové spracovanie a distribúcia na viaceré jadrá procesoru. Avšak kvôli požiadavkám na zníženie latencie, bolo potrebné jednoduché použitie *microflow cache* prehodnotiť.

Megaflow cache Výkon *microflow cache* pre krátko trvajúce spojenia a teda krátke toky bol nedostatočný. Preto bola pridaná ďalšia úroveň vyrovnávacej pamäte – *megaflow cache*. *Megaflow cache* je vyhľadávacia tabuľka, ktorá podporuje generické hľadanie zhody. Oproti OpenFlow tabuľke nepodporuje priority. Vyhľadávanie v *megaflow cache* končí pri nájdení zhody a nevyhľadáva ďalej zhodu, ktorá by mohla mať väčšiu prioritu tak, ako je to v OpenFlow tabuľke. Do *megaflow cache* sú preto inštalované iba toky, ktorých zhody sa neprelínajú. V *megaflow cache* je iba jeden klasifikátor narozdiel od zreťazenej linky v OpenFlow, preto sú sem inštalované záznamy, v ktorých sú zložené akcie zo všetkých relevantných OpenFlow tabuliek.

Cena vyhľadávania v *megaflow cache* je väčšia ako pri vyhľadávaní v *microflow cache*. Pre každý jedinečný záznam v tabuľke tokov existuje v klasifikátore jedna hešovacia tabuľka. Vyhľadávanie v *megaflow cache* znamená prehľadávanie všetkých jej hešovacích tabuliek, do kým sa nenájde zhoda. V OVS ostala zachovaná aj *microflow cache*, ktorá slúži ako prvá úroveň vyrovnávacej pamäte, v ktorej sa vyhľadáva ešte pred *megaflow cache*. *Microflow cache* je hešovacia tabuľka, ktorá určuje príslušnú tabuľku *megaflow cache*. Prvý paket toku teda prechádza cez celú *megaflow cache*, ale každý ďalší je už nasmerovaný na správnu hešovaciu tabuľku.

[5] *upstream*
[6] *flow setup time*
[7] *cache miss*

Megaflow cache obsahuje len podmnožinu všetkých OpenFlow tabuliek. OVS démon v užívateľskom priestore reaktívne počíta záznamy, ktoré sú relevantné pre súčasnú dátovú prevádzku a podľa toho pridáva do *megaflow cache* nové a odoberá staré záznamy.

2.3 Metodika testovania výkonnosti OVS

Testovanie a vyhodnocovanie výkonnosti virtuálnych prepínačov je dôležitou súčasťou ich vývoja. Pre zjednotenie výsledkov testovania vznikol automatizovaný testovací framework VSPERF [29], ktorý poskytuje obsiahlu sadu testov pre meranie výkonnosti dátovej cesty s fyzickými aj virtuálnymi rozhraniami. VSPERF umožňuje vyhodnotenie a optimalizovanie dátovej cesty pri vývoji virtuálnych prepínačov a technológií na nich založených. Architektúra frameworku podporuje použitie rôznych generátorov dátovej prevádzky a rôzne testované prepínače. Podporované sú napríklad hardvérové generátory IXIA, Spirent, Xena a softvérové generátory Moongen a T-Rex.

Odporúčania pre testovanie výkonnosti virtuálnych prepínačov, na ktorých je založený framework VSPERF, popisuje RFC 8204 [38]. RFC 8204 je zamerané predovšetkým na nastavenia testov a konfiguračných parametrov testovaného systému. Využíva pojmy a metriky definované v RFC 1242 [4], RFC 2544 [5] a RFC 2889 [19], ktoré popisujú metodiku testovania výkonnosti sieťových zariadení.

Pri meraní výkonnosti virtuálneho prepínača je dôležitých veľa faktorov, ktoré môžu ovplyvňovať konzistenciu a opakovateľnosť nameraných výsledkov. Preto je dôležité zaznamenať rôzne parametre a informácie, ktoré môžu mať významný vplyv na výsledky. RFC 8204 obsahuje zoznam parametrov, ktoré považuje za dostatočné pre zabezpečenie opakovateľnosti výsledkov. Parametre zahŕňajú informácie o použitom hardvéri, softvéri a testovacej dátovej prevádzke.

Virtuálne prepínače môžu byť v sieti nasadené v rôznych scenároch. Na obrázku 2.6 sú zobrazené scenáre nasadenia pre meranie výkonnosti, ktoré definuje RFC 8204. Nasledujúci zoznam popisuje tok dát v jednotlivých scenároch:

a) fyzický port → virtuálny prepínač → fyzický port (P2P)

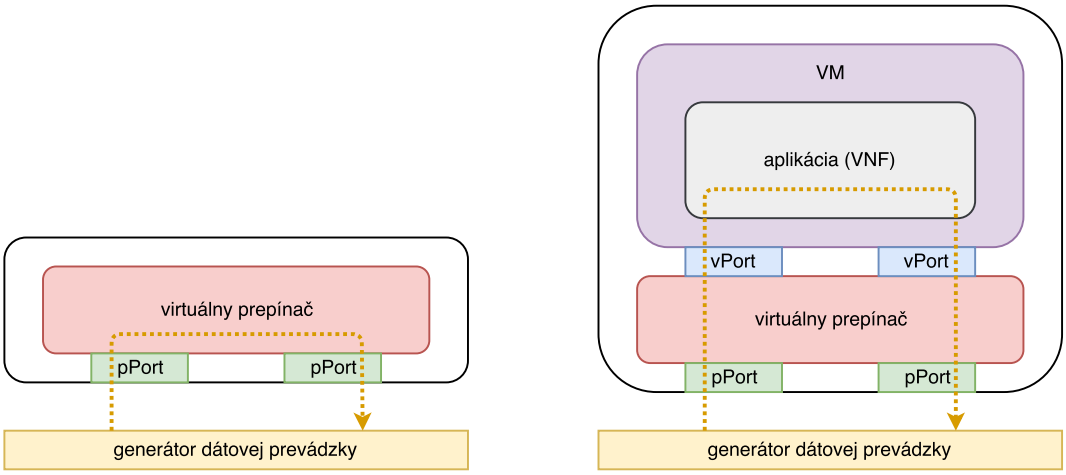
b) fyzický port → virtuálny prepínač → VNF → virtuálny prepínač → fyzický port (PVP)

c) fyzický port → virtuálny prepínač → VNF → virtuálny prepínač → VNF → virtuálny prepínač → fyzický port (PVVP)

d) fyzický port → virtuálny prepínač → VNF (P2V)

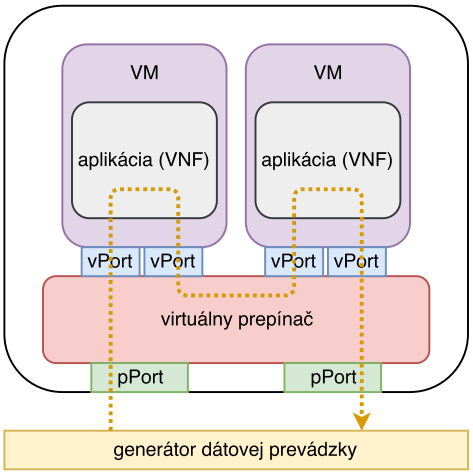
e) VNF → virtuálny prepínač → fyzický port (V2P)

f) VNF → virtuálny prepínač → VNF (V2V)

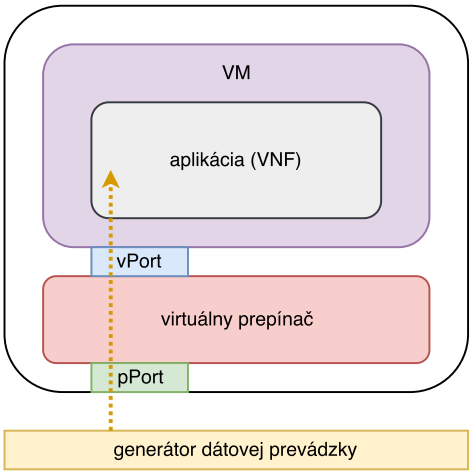


(a) P2P

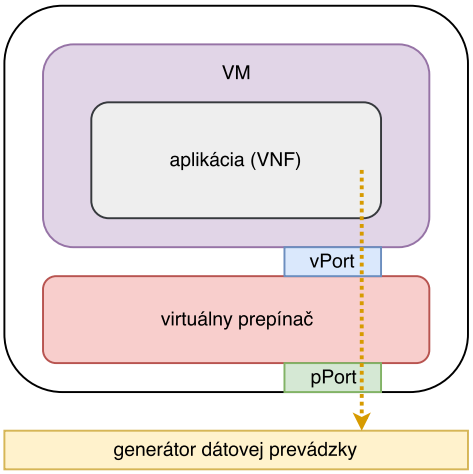
(b) PVP



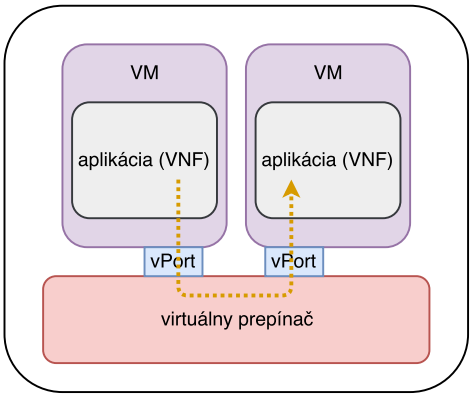
(c) PVVP



(d) P2V



(e) V2P



(f) V2V

Obr. 2.6: Schémy scenárov pre meranie výkonnosti virtuálnych prepínačov. Žltá bodkovaná čiara znázorňuje tok paketov.

Kapitola 3

Akcelerácia OVS

Klasické OVS nedokáže uspokojiť súčasné požiadavky na priepustnosť. Podľa meraní, ktoré robili Emmerich et al. [14, 13], dokáže OVS v najjednoduchšej P2P topológii spracovať 1,88 miliónov paketov za sekundu[1] (Mpps) na jednom jadre CPU. Pri ideálnom škálovaní by takto bolo potrebných 10 jadier CPU na zvládnutie plnej priepustnosti linky s rýchlosťou 10 Gb/s pri najkratších paketoch nehovoriac o linkách s rýchlosťami 40 a 100 Gb/s, ktoré sa stávajú štandardom v dátových centrách. Z tohto dôvodu vznikajú rôzne snahy o akceleráciu OVS. Akcelerácia je možná na úrovni softvéru ale aj prenesením určitej záťaže spracovania do hardvéru[2].

Akcelerácia OVS v softvéri sa dosahuje použitím frameworkov na spracovanie paketov v užívateľskom priestore operačného systému, z ktorých najznámejší a aktuálne podporovaný v hlavnej vývojovej vetve OVS je Data Plane Development Kit (DPDK). V sekcii 3.1 sú základné informácie o DPDK a v sekcii 3.2 je popísaný princíp použitia DPDK v OVS a porovnanie výkonnosti oproti klasickému OVS.

K akcelerácii OVS v hardvéri je možné zvoliť rôzne prístupy. V sekcii 3.3 sú popísané riešenia, ktoré používajú na svojich zariadeniach spoločnosti Cavium, Broadcom, Mellanox Technologies a Netronome.

3.1 DPDK

DPDK je *open source* framework pod BSD licenciou slúžiaci na rýchle spracovanie paketov v užívateľskom priestore operačného systému. Projekt bol založený Intelom a v súčasnosti podporuje viaceré architektúry procesorov: Intel x86, IBM POWER, ARM [42]. DPDK využíva viaceré techniky, ktoré umožňujú dosahovať zvýšenie rýchlosti spracovania paketov.

3.1.1 Princípy

Adresový priestor zariadenia je prostredníctvom kernel modulov UIO alebo VFIO[3] mapovaný do pamäťového priestoru užívateľskej DPDK aplikácie, čo jej umožňuje komunikovať priamo so sieťovým zariadením. Súčasťou DPDK sú vlastné buffre pre ukladanie paketov a správca pamäte pre správu týchto buffrov. Pri inicializácii aplikácie sú buffre alokované od operačného systému a počas behu sú spravované vo vlastnej réžii. Pakety sú prenášané

[1] *millions of packets per second*

[2] *hardware offload*

[3] Výnimkou sú sieťové karty od Mellanoxu a karty COMBO, ktorých ovládače v DPDK používajú vlastné kernel moduly.

medzi DPDK aplikáciou a sieťovým zariadením DMA prenosmi priamo z a do mbufov bez priechodu komplexným sieťovým protokolovým stackom jadra, ktorý môže byť pre danú aplikáciu príliš zložitý. Aplikácia tak môže byť úzko špecializovaná a efektívnejšia.

Počas priechodu spracovaním si jednotlivé komponenty DPDK aplikácie predávajú ukazovatele na mbufy, nie celé pakety. Pri potrebe replikovať pakety je možné vytvárať aj nepriame mbufy. Pri použití nepriamych mbufov nie je potrebné kopírovať paketové dáta. Funkcie pre príjem a odosielanie dát pracujú so zhlukmi paketov. Spracovanie zhluku paketov umožňuje rozložiť réžiu medzi viac paketov. Avšak čím väčšie zhluky sa používajú, tým väčšia je latencia. Preto je potrebné voliť vhodnú veľkosť zhlukov. DPDK podporuje použitie viacerých prijímacích (RX) a vysielacích (TX) front. Operácie pre príjem a odosielanie paketov nepoužívajú zámky. API zakazuje prístup viacerých jadier CPU k rovnakej fronte. Pre príjem a odosielanie paketov sa nepoužívajú prerušenia, ale aplikácia periodicky žiada o prijatie, resp. odoslanie paketov (tzv. *polling*). Takto je možné znížiť réžiu potrebnú na obsluhu prerušenia za cenu zvýšenia zaťaženia procesoru.

DPDK poskytuje API pre použitie špeciálnych inštrukcií procesoru umožňujúcich prednačítať dáta z hlavnej pamäte do cache pamäte. Táto technika sa nazýva *prefetch*. Správnym použitím v aplikácii je možné zvýšiť počet zásahov do cache[4], vďaka čomu môže stúpnuť výkon aplikácie. DPDK umožňuje pripútať vykonávanie vlákien na konkrétne jadrá CPU a taktiež alokovať pamäť s ohľadom na UMA, resp. NUMA[5] architektúru. Použitím obrovských stránok (*huge pages*) je možné znížiť réžiu prekladu virtuálnych adries na fyzické úspešnejším vyhľadávaním v TLB[6].

3.1.2 Knižnice

DPDK sa skladá z viacerých knižníc a niekoľkých typov ovládačov, ktoré sú podrobne popísané v dokumentácii [46, 44].

Environment Abstraction Layer (EAL) zapúzdruje špecifiká rôznych architektúr a sprístupňuje jednotným spôsobom nízkoúrovňové prostriedky ostatným knižniciam a aplikáciám. Medzi funkcie, ktoré zabezpečuje EAL, patrí napríklad inicializácia aplikácie, pripútanie vlákien k vykonávaniu na konkrétnych jadrách CPU, rezervácia fyzickej pamäte, abstrakcia zberníc so zariadeniami a ďalšie.

Mempool je prideľovač pamäte pre objekty pevnej dĺžky identifikovaný menom. Štandardný mempool používa pre správu objektov štruktúry `rte_ring`. Štruktúra `rte_ring` je bezzámková FIFO[7] fronta s pevným maximálnym počtom prvkov, ktorá podporuje viacerých konzumentov a producentov. Mempool poskytuje aj lokálnu vyrovnávaciu pamäť pre jednotlivé jadrá a zarovnanie objektov v pamäti s ohľadom na pamäťové kanály. V DPDK je mempool použitý pre správu mbufov a pre logovaciu službu.

Pakety v DPDK aplikáciách sú ukladané do štruktúry nazývanej **Message Buffer** (mbuf). Mbufy sú objekty s pevnou veľkosťou vytvorené pri inicializácii aplikácie a následne spravované mempoolom. Mbuf obsahuje na začiatku štruktúru `rte_mbuf`, ktorá obsahuje rôzne meta informácie. Táto štruktúra zaberá 2 riadky cache. Bezprostredne za štruktúrou `rte_mbuf` nasleduje miesto pre súkromné dáta mbufu a za nimi buffer pre objekt uložený v mbufe. Na začiatku buffru pre objekt je vyčlenené prázdne miesto, pre prípadné pripájanie hlavičiek. Ak je paket väčší ako voľné miesto v mbufe, je možné mbufy zreťaziť.

[4] *cache hit*
[5] UMA – Uniform Memory Access, NUMA – Non-Uniform Memory Access
[6] *Translation Lookaside Buffer*
[7] *First In First Out*

DPDK aplikácia komunikuje so zariadeniami prostredníctvom ovládačov[8] v užívateľskom priestore. Ovládače pre sieťové zariadenia implementujú API, ktoré sprístupňuje operácie na prácu so sieťovým zariadením ako napríklad príjem a odosielanie dát, inicializácia a konfigurácia zariadenia.

Knižnica **vhost** implementuje v užívateľskom priestore server pre sieťové zariadenie *virtio*, čo umožňuje užívateľovi posielať pakety do a prijímať pakety zo sieťového zariadenia *virtio* vo virtuálnom stroji. Nad knižnicou je postavený aj ovládač sieťového zariadenia `vhost`, ktorý ju obaluje a umožňuje posielať a prijímať pakety medzi fyzickým a virtuálnym strojom pomocou API pre sieťové zariadenia v DPDK. Ovládač `vhost` beží na strane fyzického hostiteľského systému. Pre rýchlu komunikáciu zo strany virtuálneho stroja sa používa ovládač `virtio`, ktorý beží na strane virtuálneho stroja.

3.1.3 Rozhranie rte_flow

API *rte_flow* je súčasťou rozhrania ovládačov pre sieťové zariadenia. Poskytuje prostriedky pre nastavenie a konfiguráciu filtračných schopností hardvéru. Umožňuje konfigurovať pravidlá pre hľadanie zhôd v paketoch na vstupných a výstupných rozhraniach a vykonávať operácie nad danými paketmi.

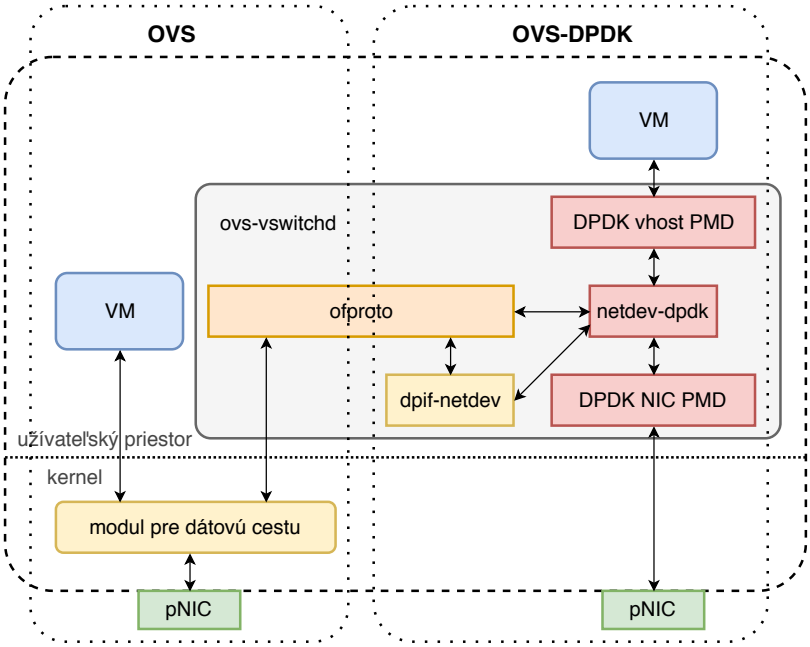
Hlavným pilierom API *rte_flow* sú pravidlá tokov. Pravidlo sa skladá z atribútov (*attributes*), vzoru (*matching pattern*) a zoznamu akcií (*actions*). Atribúty umožňujú nastaviť pra pravidlo skupinu, prioritu a smer (*ingress* alebo *egress*). Podporované sú viaceré skupiny a úrovne priorít, ale ich implementácia v ovládači nie je vyžadovaná. Vzor sa skladá z položiek (*item*), ktoré sa dajú rozdeliť do dvoch kategórií: položky určujúce protokolové hlavičky a dáta; meta položky určujúce meta dáta alebo priamo ovplyvňujúce proces určovania zhody. Položky sú vo vzore uložené v zozname od najnižšej protokolovej vrstvy po najvyššiu. Poradie položiek popisujúcich protokolové hlavičky je teda dôležité. Meta položky sa môžu nachádzať v zozname kdekoľvek bez vplyvu na výsledok. Položka umožňuje definovať presnú zhodu, rozsah hodnôt alebo masku pre políčka hlavičky. Pravidlo môže obsahovať zoznam akcií, ktoré sú vykonávané v poradí. Akcie je možné rozdeliť do troch kategórií: akcie určujúce osud paketov vyhovujúcich vzoru (napríklad zahodenie, poslanie do fronty); akcie upravujúce pakety vyhovujúce vzoru (napríklad pridávanie, odoberanie hlavičiek, šifrovanie, kompresia, značkovanie); akcie týkajúce sa samotného pravidla (napríklad úprava čítačov). Kompletný zoznam aktuálne podporovaných položiek a akcií je možné nájsť v dokumentácii pre API *rte_flow* [45].

Správa pravidiel je zabezpečená niekoľkými funkciami: validácia pravidla, vytvorenie pravidla, zrušenie pravidla, zrušenie všetkých pravidiel, dopyt na pravidlo (získanie dát špecifických pre tok ako napríklad hodnotu čítačov), zapnutie/vypnutie izolovaného režimu. Izolovaný režim znamená, že iba pakety vyhovujúce pravidlám sa dostanú do vstupných (RX) front. Bez izolovaného režimu sa do vstupných front môžu dostať aj pakety, pre ktoré nevyhovuje žiadne pravidlo.

3.2 OVS-DPDK

Dátová cesta OVS je štandardne implementovaná v module jadra. OVS ale podporuje aj konfiguráciu, kedy je dátová cesta spracovávaná v užívateľskom priestore a teda aj celé OVS môže bežať len v užívateľskom priestore. Pre tieto účely je implementovaný *dpif provider*

[8] *Poll Mode Driver* (PMD)



Obr. 3.1: Porovnanie architektúr OVS a OVS využívajúceho DPDK (OVS-DPDK).

`dpif-netdev` (kapitola 2, sekcia 2.2). Pakety do dátovej cesty v užívateľskom priestore prichádzajú cez `netdev`. Implementácia využívajúca štandardné linuxové rozhranie je pomalá. Zaujímavá je však implementácia `netdev-dpdk` umožňujúca prenosy dát cez DPDK, s ktorou je možné dosiahnuť lepšiu výkonnosť ako s dátovou cestou v jadre.

Obrázok 3.1 zobrazuje porovnanie architektúry OVS s dátovou cestou v module jadra a s dátovou cestou v užívateľskom priestore (`dpif-netdev`) s využitím DPDK (OVS-DPDK). *Netdev provider* `netdev-dpdk` používa DPDK PMD pre fyzickú sieťovú kartu pre príjem a odosielanie paketov zo sieťovej karty a DPDK vhost PMD pre presun paketov do a z virtuálneho stroja. Obsluhu sieťových rozhraní v OVS-DPDK zabezpečujú tzv. *PMD vlákna*[9], ktoré sú popísané v sekcii 3.2.1. V dátovej ceste v užívateľskom priestore je tiež implementovaný systém vyrovnávacích pamätí pre toky podobne ako pre dátovú cestu v jadre (sekcia 2.2.3). Systém vyrovnávacích pamätí pre toky v OVS-DPDK je popísaný v sekcii 3.2.2. Priepustnosť OVS a OVS-DPDK je porovnaná v sekcii 3.2.3.

3.2.1 PMD vlákna

PMD vlákna zabezpečujú pre dátovú cestu v OVS-DPDK odoberanie paketov z prijímacích (RX) front vstupných portov, klasifikovanie paketov a vykonanie akcií, medzi ktoré môže patriť napríklad odoslanie paketu na výstupnú (TX) frontu výstupného portu. Vo východzom stave je vytvorené jedno PMD vlákno pre každý NUMA uzol, na ktorom existuje aspoň jedno DPDK rozhranie pridané do OVS. Konfiguračné možnosti OVS umožňujú vytvoriť aj viac PMD vlákien, na konkrétnych jadrách CPU. Napríklad vytvorenie štyroch PMD vlákien na jadrách 0, 1, 2 a 3 sa nastaví príkazom:

```
ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0xf
```

[9] *Poll Mode Driver threads*

Priradenie RX front na jednotlivé PMD vlákna môže byť nastavené manuálne užívateľom alebo automaticky. Užívateľ môže pre každý port pridaný do OVS nastaviť zoznam, ktorý obsahuje dvojice identifikačných čísel RX fronty a jadra CPU. RX fronta môže byť priradená iba na jadro CPU, pre ktoré bolo vytvorené PMD vlákno pomocou konfiguračnej voľby `pmd-cpu-mask`. Napríklad priradenie štyroch RX front 0, 1, 2 a 3 portu `dpdk0` postupne na jadrá 0, 1, 2 a 3 sa vykoná príkazom:

```
ovs-vsctl set Interface dpdk0 options:n_rxq=4 \
    other_config:pmd-rxq-affinity="0:0,1:1,2:2,3:3"
```

PMD vlákna, ktoré bežia na jadrách, ku ktorým bola priradená aspoň jedna RX fronta, sa stávajú izolovanými. V izolovaných PMD vláknach sú obsluhované iba RX fronty, ktoré boli na dané jadro manuálne priradené. Pre RX fronty, ktorým nebolo priradené jadro CPU manuálne, je vybrané PMD automaticky na základe štatistiky počtu cyklov CPU pri spracovaní, ktorú majú RX fronty uloženú.

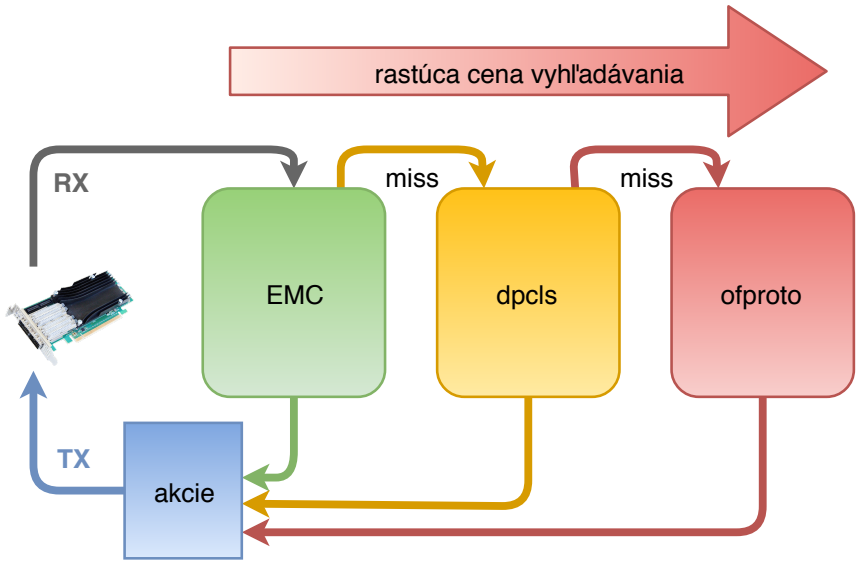
Priradenie TX front je vykonávané automaticky. PMD vlákno používa jednu TX frontu pre každý port, na ktorý potrebuje odosielať pakety. Ak má port počet TX front väčší alebo rovný počtu PMD vlákien, tak sú TX fronty priradené PMD vláknam sekvenčne podľa identifikačného čísla fronty a vlákna začínajúc od najnižších čísel. Ak port nemá dostatočný počet TX front, tak sú TX fronty prideľované PMD vláknam mechanizmom nazývaným *Transmit Packet Steering* (XPS). XPS prideľuje čísla TX front PMD vláknam dynamicky. Ak je TX fronta pridelená PMD vláknu určitý čas nepoužívaná, tak je z daného PMD vlákna uvoľnená. Keď potrebuje PMD vlákno poslať pakety na port, ku ktorému nemá pridelenú TX frontu, nechá si prideliť TX frontu z daného portu. Port nesie referenčné čítače počtu vlákien používajúcich TX fronty, podľa ktorých je vybraná najmenej používaná TX fronta.

3.2.2 Vyrovnávacie pamäte pre toky v OVS-DPDK

OVS je virtuálny prepínač podporujúci protokol OpenFlow, pomocou ktorého je možné vykonávať aj pokročilé sieťové funkcie, čo môže vo výsledku vytvárať zložité a dlhé linky pre spracovanie paketov. Aby bolo možné spracovávať pakety pri vysokých rýchlostiach, implementované sú viaceré úrovne vyrovnávacích pamätí pre tabuľky tokov, ktoré uchovávajú aktívne toky.

OVS-DPDK obsahuje tri úrovne tabuliek tokov. S každou ďalšou úrovňou tabuliek sa zvyšuje doba potrebná pre klasifikáciu paketu, a teda aj určeniu príslušných akcií, ktoré sa pre daný paket majú vykonať. Tento proces je znázornený na obrázku 3.2. Prvou a najrýchlejšou úrovňou je tabuľka nazývaná *Exact Match Cache* (EMC), ktorá nepodporuje vyhľadávanie zhody pomocou masiek. Keď nie je nájdená zhoda pre paket v EMC, pokračuje sa vyhľadávaním v ďalšej tabuľke nazývanej *datapath classifier* (*dpcls*), ktorá podporuje už aj vyhľadávanie pomocou masiek. Ak ani vyhľadanie v *dpcls* nie je úspešné, paket postúpi do poslednej úrovne, ktorú tvorí *ofproto* klasifikátor, čo je implementácia tabuliek tokov pre OpenFlow. Implementácia OpenFlow tabuliek v OVS poskytuje 255 tabuliek tokov, pričom OpenFlow kontroléry môžu používať tabuľky 0 – 127 a tabuľky s číslom 128 a viac sú rezervované pre vnútorné použitie samotným prepínačom [43].

Exact Match Cache (EMC) EMC je prvá a najrýchlejšia tabuľka, s ktorou je konfrontovaný paket prichádzajúci zo sieťového rozhrania. EMC podporuje iba presné vyhľadanie zhody, žiadne masky. Pre každé PMD vlákno existuje samostatná EMC tabuľka. Podrobný popis implementácie EMC je v článku na webovej stránke Intelu v zóne pre vývojárov [26].



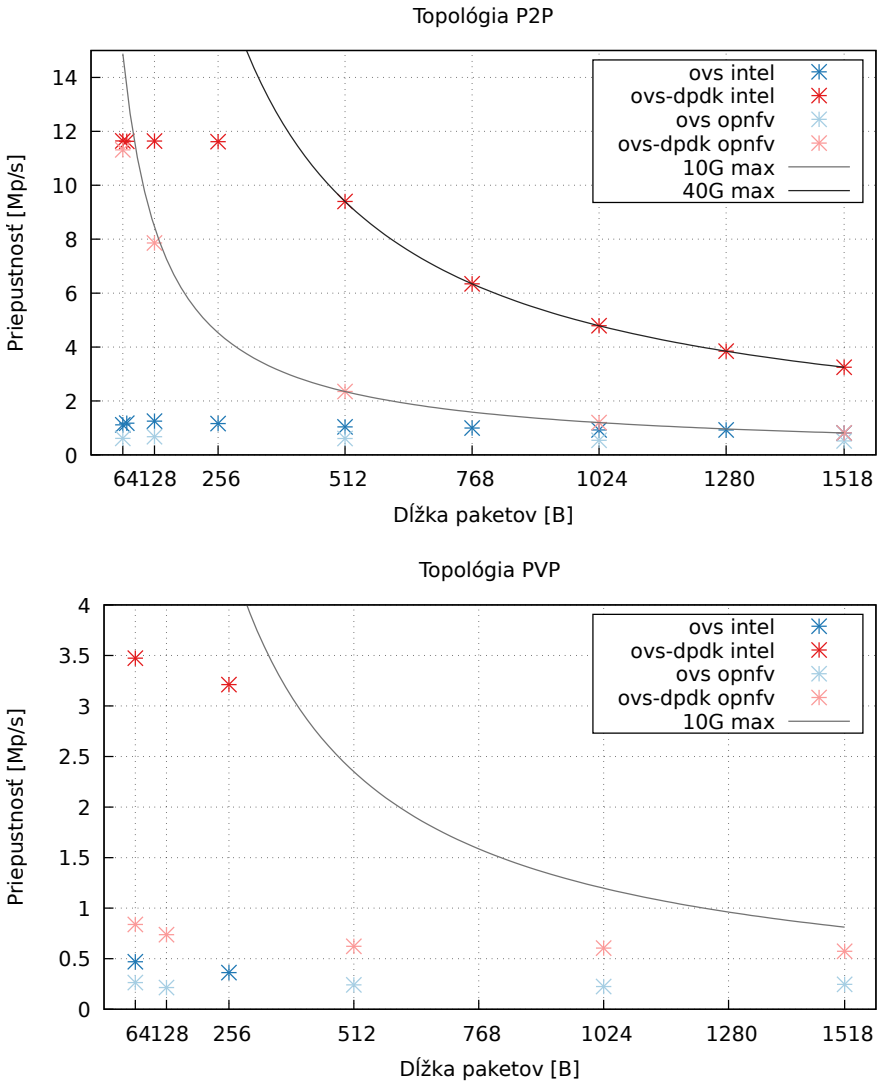
Obr. 3.2: Hierarchia vyrovnávacích pamätí pre toky v OVS-DPDK tvorená tromi úrovňami tabuliek. Prvú úroveň tvorí *Exact Match Cache* (EMC), druhú úroveň tzv. *datapath classifier* (*dpcls*) a poslednú úroveň OpenFlow tabuľky (*ofproto*). S každou ďalšou úrovňou tabuliek rastie cena vyhľadávania akcií pre spracovávaný paket.

Datapath classifier (dpcls) Pri reálnom nasadení virtuálneho prepínača prichádzajú do prepínača tisícky tokov, čo znamená, že EMC sa rýchlo zaplní. Preto je veľmi dôležitá výkonná druhá úroveň tabuliek, ktorú tvorí *dpcls* klasifikátor. Každé PMD vlákno má vlastný *dpcls* klasifikátor, rovnako ako aj EMC. *Dpcls* využíva algoritmus *Tuple Space Search* [37] a umožňuje tak vyhľadávanie s použitím masiek. V článkoch v zóne pre vývojárov na webovej stránke Intelu je podrobne popísaný princíp [2] a implementácia [3] *dpcls*.

3.2.3 Porovnanie výkonnosti OVS a OVS-DPDK

Na obrázku 3.3 sú pre porovnanie ukázané grafy priepustnosti štandardného OVS a OVS s DPDK na topológiách P2P a PVP pre jedno jadro CPU. Hodnoty sú prevzaté zo správy o výsledkoch meraní od OPNFV s platformou VSPERF [28] (v grafoch označené ako *opnfv*) a zo správy o testoch OVS od spoločnosti Intel [17] (v grafoch označené ako *intel*). V oboch prípadoch bol použitý rozdielny hardvér aj verzie softvéru, čo má výrazný vplyv na výkon.

V meraniach od OPNFV boli použité dva 10G porty sieťového adaptéru 82599ES od Intelu v oboch smeroch, čo umožňuje vo výsledku dosiahnuť maximálnu priepustnosť 20 Gb/s. Pre obsluhu sieťových rozhraní v OVS boli použité dve jadrá CPU. S topológiou P2P jedno jadro pre každé fyzické rozhranie. Pri topológii PVP boli použité dva virtuálne stroje, každý s jedným virtuálnym rozhraním. Každá dvojica fyzické a virtuálne rozhranie mala pridelené jedno jadro CPU. V meraniach od Intelu boli použité dve dvojice 10G portov dvoch sieťových adaptérov X710-DA4 v oboch smeroch, a teda celková maximálna priepustnosť bola 40 Gb/s. Pri topológii PVP bol použitý jeden virtuálny stroj so štyrmi virtuálnymi rozhraniami. Všetky sieťové rozhrania v OVS boli obsluhované jedným jadrom CPU. V oboch meraniach bola na preposielanie paketov vo virtuálnom stroji použitá DPDK aplikácia `testpmd` a virtuálne stroje bežali na iných jadrách CPU ako OVS. Hodnoty z meraní od OPNFV boli prepočítané na jedno jadro CPU (vydelené dvomi).



Obr. 3.3: Porovnanie priepustnosti OVS a OVS-DPDK na jednom jadre CPU pre topológie P2P a PVP. Dáta sú prevzaté zo správ o výsledkoch meraní od OPNFV [28] (označené *opnfv*) a od Intelu [17] (označené *intel*).

Použitie OVS s DPDK umožňuje výrazné zvýšenie priepustnosti, avšak každé vlákno zabezpečujúce prijímanie a odosielanie paketov využíva kvôli technike pollingu jadro CPU na 100%. Pri topológii P2P bola dosahovaná približne 10 – 18-krát vyššia priepustnosť ako s OVS bez DPDK. Pri topológii PVP umožnilo použitie DPDK približne 3 – 7-krát vyššiu priepustnosť. Prijímacie a vysielacie fronty sieťových rozhraní je možné spracovávať na rôznych jadrách a dosiahnuť tak ešte vyššiu priepustnosť, ale dôsledkom je menší počet jadier dostupných pre VNF vo virtuálnom stroji.

3.3 Akcelerácia OVS v hardvéri

Keďže možnosti softvérovej akcelerácie sa blížia k svojim limitom a požiadavky na priepustnosť sa stále zvyšujú, vznikajú snahy o akceleráciu OVS v hardvéri. Priepustnosť OVS

s DPDK je síce možné zvyšovať pridávaním jadier CPU spracovávajúcich pakety, ale tým sa znižuje počet jadier dostupných pre virtualizované aplikácie. Presunutím záťaže OVS do hardvéru je možné zvýšiť priepustnosť a zároveň znížiť počet jadier CPU zabezpečujúcich prepínanie paketov medzi fyzickými a virtuálnymi rozhraniami, čo vedie k zníženiu nákladov na prevádzku virtualizovaných prostredí. O presunutie záťaže do hardvéru sa snažia viacerí výrobcovia inteligentných sieťových zariadení rôznymi technikami. Používané prístupy sú beh OVS priamo v sieťovej karte (3.3.1) a offload tabuliek tokov z OVS do hardvéru, ktorý môže byť čiastočný (3.3.2) alebo plný (3.3.3).

3.3.1 OVS v sieťovej karte

Niektoré inteligentné sieťové karty ako napríklad LiquidIO® [7] od spoločnosti Cavium alebo NetXtreme® S-Series [6] od spoločnosti Broadcom umožňujú beh OVS priamo v sieťovej karte.

V prípade kariet LiquidIO® obsahuje karta procesorové jadrá založené na báze MIPS. Časť jadier je určená pre Linux, v ktorom beží `ovsdb-server` a `ovs-vswitchd` slúžiace na správu a konfiguráciu OVS. Zvyšok jadier je použitý na akcelerované spracovanie paketov. Pakety môžu byť spracovávané pomalou alebo rýchlou cestou. Jadrá pre spracovanie paketov si udržiavajú tabuľky tokov, ktoré sú synchronizované s tabuľkami tokov v OVS bežiacom na jadrách s Linuxom. Ak sa v tabuľkách tokov nájde zhoda, vykonajú sa príslušné akcie a paket je doručený na správne miesto rýchlou cestou. Ak sa v tabuľkách zhoda nenájde, paket je presunutý do OVS a vyhladaný v jeho tabuľkách. Následne sú aktualizované tabuľky v jadrách spracovávajúcich pakety a paket je spracovaný podľa príslušných pravidiel. Jedná sa o pomalú cestu.

Beh OVS v kartách NetXtreme® S-Series umožňuje procesorový blok obsahujúci osem ARMv8 Cortex-A72 jadier, na ktorom tak môže bežať operačný systém a v ňom kompletné OVS. K akcelerovanému spracovaniu tokov v karte je použitá technológia TruFlow™.

3.3.2 Čiastočný offload tabuliek tokov do hardvéru

Čiastočný offload znamená, že časť spracovania paketov prebieha v sieťovej karte a časť v OVS bežiacom na CPU cieľového fyzického stroja. Príkladom operácií, ktoré môžu byť spracované v sieťovej karte, je klasifikácia paketov do tokov. OVS má k dispozícii rozhranie pomocou, ktorého môže nahrať do karty pravidlá určujúce toky. Pakety sú potom klasifikované v karte a akcie sú dokončené dátovou cestou v softvéri.

Príkladom kariet umožňujúcich čiastočný offload sú sieťové karty od spoločnosti Mellanox Technologies, ktoré obsahujú vstavaný programovateľný prepínač (eSwitch) implementujúci prepínanie medzi virtuálnymi sieťovými rozhraniami [20]. Vďaka tomuto vstavanému prepínaču je možné vykonať určité operácie pri spracovaní paketov priamo v karte.

Čiastočný offload do hardvéru je dosahovaný pri použití technológie *ASAP*^{2} *Flex*. V tomto modele nasadenia používajú virtuálne stroje paravirtualizáciu a pakety putujú cez virtuálny prepínač. Virtuálny prepínač môže cez API nastaviť, aby sieťová karta spracovávala určité výpočetne náročné operácie ako napríklad klasifikovanie paketov. Doručenie paketov na cieľové virtuálne rozhranie však zabezpečuje OVS, karta pakety len pripraví.

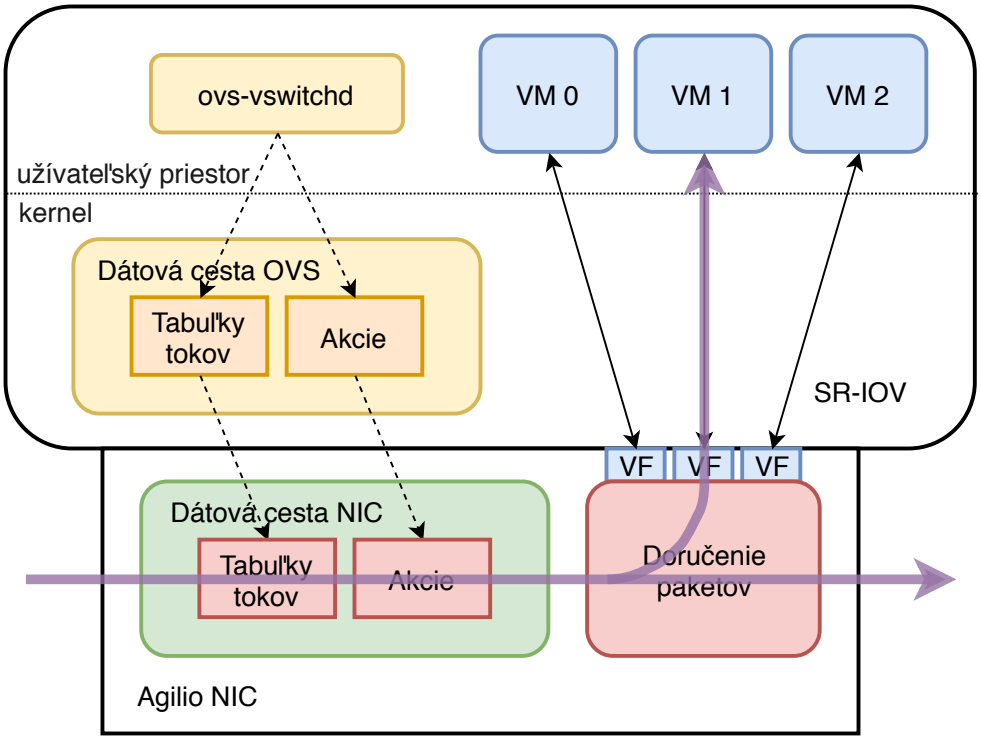
3.3.3 Plný offload tabuliek tokov do hardvéru

Pri plnom offloade dokáže sieťová karta vykonať celú dátovú cestu vrátane doručenia paketu. Doručenie paketu do virtuálneho stroja môže prebiehať cez virtuálne funkcie SR-IOV alebo virtio. Ak karta neobsahuje pravidlo pre spracovanie paketu a nedokáže daný paket spracovať, paket je poslaný do dátovej cesty v softvéri.

Sieťové karty od spoločnosti Mellanox Technologies podporujú aj plný offload pri nasadení technológie *ASAP*^{2} *Direct*, kedy sú pakety presúvané do virtuálnych strojov priamo cez virtuálne funkcie SR-IOV. Na základe užívateľsky definovaných pravidiel sú záznamy z tabuliek tokov v `ovs-vswitchd` naprogramované do sieťovej karty namiesto do modulu v jadre. Pakety patriace do tokov, pre ktoré karta obahuje pravidlá, môžu byť kompletne spracované v karte vrátane doručenia na správne cieľové rozhranie.

Netronome je ďalším výrobcom inteligentných sieťových kariet, ktorý úspešne využíva plný offload do hardvéru k akcelerácii OVS s kartami Agilio [24, 22]. V sieťových kartách Agilio sú použité programovateľné sieťové procesory[10] [25]. Netronome využíva k akcelerácii OVS dva prístupy.

Prvý prístup využíva vlastné úpravy v jadre k presunu a synchronizácii tabuliek tokov z modulu v jadre do sieťovej karty. Na obrázku 3.4 je znázornená schéma tohto spôsobu akcelerácie OVS. Hľadanie zhody v paketoch a vykonávanie akcií s paketmi sa môže uskutočňovať na karte, zatiaľ čo kontrolná časť OVS beží stále v užívateľskom priestore na serveri.

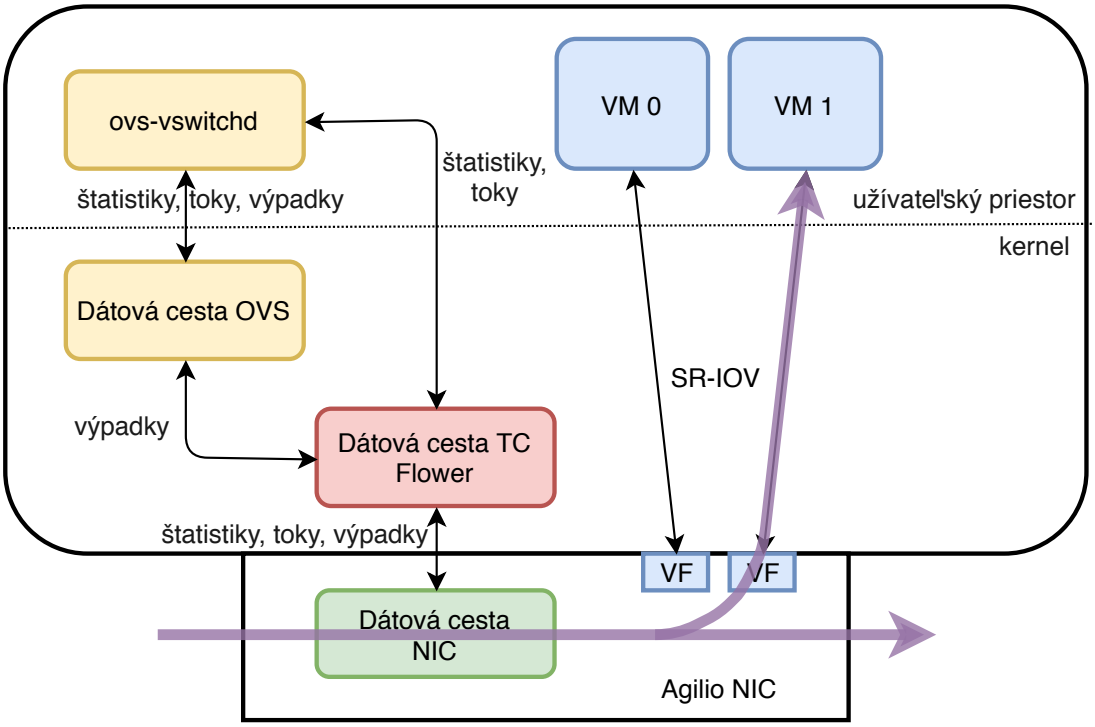


Obr. 3.4: Schéma akcelerácie OVS, pri ktorej Netronome používa vlastné úpravy v jadre, k zabezpečeniu offloadu spracovania tokov priamo v karte.

[10] *network flow processors*

Súčasťou akcelerovanej dátovej cesty v karte je tabuľka pre vyhľadávanie presnej zhody a sada hešovacích tabuliek, v ktorých sa vyhľadáva, ak sa nenájde presná zhoda. Vypĺňanie a aktualizácie tabuliek tokov v karte je zabezpečené pomocou zmien dátovej cesty v jadre. Keď sú nové záznamy tokov nahrávané démonom v užívateľskom priestore do dátovej cesty v jadre, záznamy sa zároveň nahrajú aj do karty a pri spracovaní ďalších paketov z rovnakých tokov sa uplatní vyhľadanie v karte. K presunom dát medzi kartou a virtuálnym strojom sa používajú virtuálne funkcie SR-IOV. Každá VF asociovaná s virtuálnym strojom je vo virtuálnom prepínači reprezentovaná ako virtuálny port. Pakety môžu byť potom prenášané prostredníctvom VF priamo medzi kartou a virtuálnym strojom.

Prvý prístup k akcelerácii OVS nebol podporovaný komunitou a nedostal sa do hlavnej vývojovej vetvy OVS. Netronome pracoval aj na ďalšom prístupe pre podporu offloadu do hardvéru, ktorý podporila aj komunita. V júni 2017 sa API pre podporu offloadu do hardvéru vytvorené v spolupráci s Mellanoxom stalo súčasťou hlavnej vývojovej vetvy OVS. API pridáva do knižnice `netdev` operácie, ktoré používa `dpif` vrstva.



Obr. 3.5: Schéma akcelerácie OVS s kartami Agilio od spoločnosti Netronome využívajúcej klasifikátor TC Flower v jadre Linuxu.

Komunitou podporovaný prístup k akcelerácii OVS používa okrem dátovej cesty aj ďalšiu súčasť jadra a to klasifikátor paketov pre klasifikáciu dátovej prevádzky v jadre Linuxu – TC Flower [16]. Klasifikátor TC Flower umožňuje hľadanie zhody v paketoch na dopredu definované položky, vykonávať modifikácie, preposielanie a zahadzovanie paketov. Obrázok 3.5 zobrazuje schému akcelerácie založenej na TC Flower. Pravidlá pre spracovanie tokov môžu byť pomocou TC Flower nahraté aj do hardvéru. Súčasťou konfigurácie sú príznaky, ktoré určujú či je pravidlo spracovávané v softvéri alebo v hardvéri. Netronome pri nahrávaní pravidiel do TC Flower používa príznak určujúci, že sa pravidlo nahrá len do hardvéru a teda TC Flower vo výsledku neobsahuje žiadne pravidlá. Cieľom pri offlo-

ade tokov je nahrávanie všetkých pravidiel do TC Flower a v prípade, že pravidlo nie je podporované, uloží si ho modul pre dátovú cestu.

Netronome vykonal výkonnostné testy svojho riešenia akcelerácie, ktoré porovnal s výkonom sieťovej karty XL710 od Intelu [24, 23]. Na karte od Intelu merali výkon OVS s dátovou cestou v jadre aj v užívateľskom priestore s použitím DPDK. Pri testoch použili dva procesory s 12 jadrami. Pre neakcelerované OVS na karte od Intelu potrebovali použiť 12 jadier CPU pre dátovú cestu aj v jadre aj v DPDK, aby dosiahli vyrovnanú priepustnosť v OVS aj v aplikácii. Pre virtualizované stroje tak ostalo 12 jadier. OVS akcelerované na karte Agilio potrebovalo použiť na plnú priepustnosť len 1 jadro CPU, vďaka čomu ostáva pre virtuálne stroje o 11 jadier CPU viac ako pri neakcelerovanom riešení. V tabuľke 3.1 je prehľad výsledkov, ktoré nameral Netronome. Merali jednoduché L2 preposielanie paketov s doručením do cieľového virtuálneho stroja a analýzu VXLAN hlavičiek. Tabuľka zobrazuje maximálnu dosiahnutú paketovú rýchlosť s najkratšími paketmi (64 B), ktorá sa držala stabilne aj pre väčšie pakety, kým nebola dosiahnutá plná rýchlosť linky. Ďalej sú v tabuľke uvedené veľkosti paketov, s ktorými dosiahli plnú rýchlosť linky.

Tabuľka 3.1: Maximálne paketové rýchlosti (pre 64 B pakety) a veľkosti paketov potrebné pre plnú rýchlosť 40G linky v rôznych variantách OVS s kartami Agilio od spoločnosti Netronome.

Jednoduché L2 preposielanie do VM			**Spracovanie VXLAN a posielanie do VM**		
Variant	Maximálna rýchlosť	Rýchlosť 40G linky	Variant	Maximálna rýchlosť	Rýchlosť 40G linky
OVS	5,8 Mpps	$>$ 1000 B	**OVS**	1,5 Mpps	-
OVS-DPDK	12,1 Mpps	$>$ 400 B	**OVS-DPDK**	8 Mpps	$>$ 512 B
offload OVS	28 Mpps	$>$ 160 B	**offload OVS**	22 Mpps	$>$ 200 B

Kapitola 4

Akceleračné karty COMBO

Sieťové karty COMBO vyvíjané združením CESNET sú založené na technológii čipov FPGA. Karty COMBO sú komerčne dostupné pod názvom *Netcope FPGA Board* (NFB) [21] od spoločnosti Netcope Technologies. Karty obsahujú ethernetové rozhrania podporujúce rýchlosti až do 100 Gb/s a rozhranie zbernice PCI-Express umožňujúce rýchle DMA prenosy dát medzi kartou a pamäťou hostiteľského stroja. Príkladom použitia kariet COMBO je akcelerácia monitorovania a filtrovania vo vysokorýchlostných sieťach. Flexibilita, ktorou disponujú vďaka FPGA čipu, umožňuje ich použitie v rôznych aplikáciách, v ktorých je potrebné zabezpečiť vysokú priepustnosť dát do softvéru alebo rýchle spracovanie s nízkou odozvou priamo v hardvéri. 100G Ethernet podporujú karty COMBO-100G1, COMBO-100G2Q, COMBO-100G2C a najnovšia nízkoprofilová karta COMBO-200G2QL. Karty obsahujú FPGA čipy od Xilinxu.

Karta COMBO-200G2QL je bližšie popísaná v sekcii 4.1. Karty COMBO obsahujú proprietárne rozhranie pre DMA prenosy po zbernici PCI-Express, ktoré je popísané v sekcii 4.2. Vďaka FPGA čipu na karte je funkcionalita karty určená firmvérom, ktorý je aktuálne nahraný v FPGA. Nahrať nový firmvér je možné v priebehu niekoľkých sekúnd. V sekcii 4.3 je popísaný jazyk P4, ktorý umožňuje definovať spracovanie paketov sieťovým zariadením. Z popisu v jazyku P4 je následne možné vygenerovať firmvér do FPGA obsahujúci definovaný analyzátor paketov.

4.1 COMBO-200G2QL

COMBO-200G2QL je nízkoprofilová inteligentná sieťová karta. Karta obsahuje dve sieťové rozhrania QSFP28 a podporuje ethernetové štandardy 100GBASE-SR4 / LR4, 25G / 50GBASE-SR, 40GBASE-SR4 / LR4 / IR4 a 10GBASE-SR / LR. Podporované konfigurácie rýchlostí rozhraní sú 2x 100G, 4x 50G, 2x 40G, 8x 25G a 8x 10G.

Jadrom karty je FPGA čip UltraScale+ od Xilinxu, ktorý podporuje PCI-Express bloky pre 16 liniek (x16), narozdiel od predchádzajúceho čipu Virtex-7 H580T nachádzajúceho sa na starších kartách COMBO-100G1, COMBO-100G2Q, COMBO-100G2C, ktorý podporuje len PCI-Express bloky x8. Karta COMBO-200G2QL obsahuje dve rozhrania PCI-Express generácie 3 so 16 linkami. Každé rozhranie umožňuje maximálnu teoretickú priepustnosť 128 Gb/s. Celková teoretická priepustnosť je 128 + 128 Gb/s. Do hostiteľského stroja môže byť pripojené jedno alebo obe PCI-Express x16 rozhrania. Karta podporuje pripojenie do systému s dvomi CPU, pričom každé PCI-Express rozhranie môže byť pripojené k inému CPU.

Obr. 4.1: Nízkoprofilová karta COMBO-200G2QL. Obrázok prevzatý z [21].

Karta COMBO-200G2QL obsahuje aj statické RAM pamäte QDRIIIe vo variante s tromi modulmi s kapacitou 72 Mb alebo tri moduly s kapacitou 288 Mb.

Firmvér karty v FPGA sa skladá zo základných komponent spoločných pre rôzne firmvéry a aplikačného jadra, ktoré určuje funkcionalitu karty. Medzi dôležité základné bloky patria: sieťový modul pre obsluhu ethernetových rozhraní, radič pamätí, DMA radič riadiaci činnosť DMA prenosov a ďalšie podporné komponenty. Firmvér pre kartu COMBO-200G2QL obsahuje dva DMA radiče. Každý využíva jedno PCI-Express rozhranie. Prostredníctvom rýchleho dátového rozhrania popísaného v sekcii 4.2 je v kombinácii s obomi DMA radičmi možné prenášať do pamäte RAM hostiteľského stroja dáta pri plnej rýchlosti oboch sieťových 100G rozhraní.

4.2 Dátové rozhranie

Pre rýchle DMA prenosy dát medzi kartou a pamäťou RAM v počítači po zbernici PCI-Express slúži rozhranie SZE2[1]. Na strane operačného systému je rozhranie SZE2 tvorené systémovými ovládačmi. Ovládače zabezpečujú inicializáciu, konfiguráciu, riadenie karty a prípravu buffrov v pamäti RAM pre DMA prenosy. DMA prenosy sú riadené DMA radičom implementovaným v FPGA na karte. Rozhranie SZE2 sa skladá z niekoľkých samostatných kanálov pre prijímací (RX) aj vysielací (TX) smer. Počet kanálov a distribúcia paketov medzi kanály závisí od použitého firmvéru.

Každý kanál obsahuje kruhový buffer v pamäti RAM a v pamäti na karte pre každý smer. Buffer v pamäti RAM sa skladá z niekoľkých veľkých súvislých pamäťových blokov, ktoré sú odkazované deskriptormi a pomocou týchto deskriptorov prepojené do kruhovej štruktúry. Počet a veľkosť blokov je nastaviteľná pomocou parametrov modulu jadra. Pre každý buffer existuje dvojica ukazovateľov na začiatok a koniec dát v buffri. DMA prenosy sú iniciované a riadené DMA radičom na základe hodnôt týchto ukazovateľov.

[1] *Straight ZEro copy* druhá verzia 2

Rozhranie SZE2 bolo navrhnuté s ohľadom na prenos segmentov dát rôznych dĺžok, čomu zodpovedá aj formát prenášaných dát. SZE2 segment obsahuje hlavičku a dáta. Hlavička nesie informáciu o dĺžke segmentu prípadne dodatočné informácie z akceleračného aplikačného jadra karty. Dáta obsahujú rámec zo spojovej vrstvy. Hlavička aj dáta sú zarovnané na 8 bytov. Pakety sú v buffri umiestnené súvisle za sebou. Voľné miesto vzniká len v dôsledku zarovnania.

Aplikáciám v užívateľskom priestore sprístupňuje rozhranie SZE2 API knižnice libsze2. Knižnica libsze2 zapúzdruje rozhranie modulov jadra a poskytuje funkcie pre jednoduchú inicializáciu DMA kanálov, spúšťanie a zastavovanie DMA prenosov, prijímanie a odosielanie paketov.

Rozhranie SZE2 je navrhnuté pre prúdové spracovanie dát. Model spracovania paketov počíta so spracovaním paketov jeden za druhým. Využíva sa princíp *zero copy*, kedy je buffer z jadra namapovaný do užívateľského priestoru a aplikácii je poskytovaný ukazovateľ priamo na dáta v buffri. V RX smere sa po spracovaní paketu posúva ukazovateľ ohraničujúci obsadené a voľné miesto v buffri a do voľnej pamäte je tak možné preniesť ďalšie pakety. V TX smere aplikácia žiada ovládač o voľné miesto v buffri, do ktorého zapíše pakety, ktoré sa majú odoslať do siete. Ovládač sa potom v spolupráci s DMA radičom postará o odoslanie paketov a uvoľnenie miesta pre zápis ďalších dát.

4.3 Jazyk P4

Programming Protocol-independent Packet Processors (P4) [50] je vysokoúrovňový jazyk pre popis spôsobu spracovania paketov dátovou cestou programovateľných prepínacích sieťových prvkov. Cieľom jazyka je poskytovať prostriedky pre definovanie funkcionality sieťových zariadení nezávisle na použitých protokoloch a cieľovej platforme. Jazyk P4 je založený na abstraktnom modele, ktorý sa skladá z analyzátoru a množiny tzv. *match+action* tabuliek rozdelených medzi vstupom a výstupom. Analyzátor identifikuje hlavičky nachádzajúce sa v každom prichádzajúcom pakete. *Match+action* tabuľky uskutočňujú vyhľadávanie na podmnožine políčok hlavičiek a vykonávajú akcie prislúchajúce prvej nájdenej zhode[2].

Jazyk P4 umožňuje definovať spracovanie paketov sieťového zariadenia pomocou nasledujúcich základných prvkov:

Definície hlavičiek – popisujú hlavičky protokolov, ktoré bude sieťové zariadenie podporovať.

Analyzátor paketu – popisuje konečný automat použitý k analýze paketu, pri ktorej sa extrahujú hodnoty políčok hlavičiek protokolov.

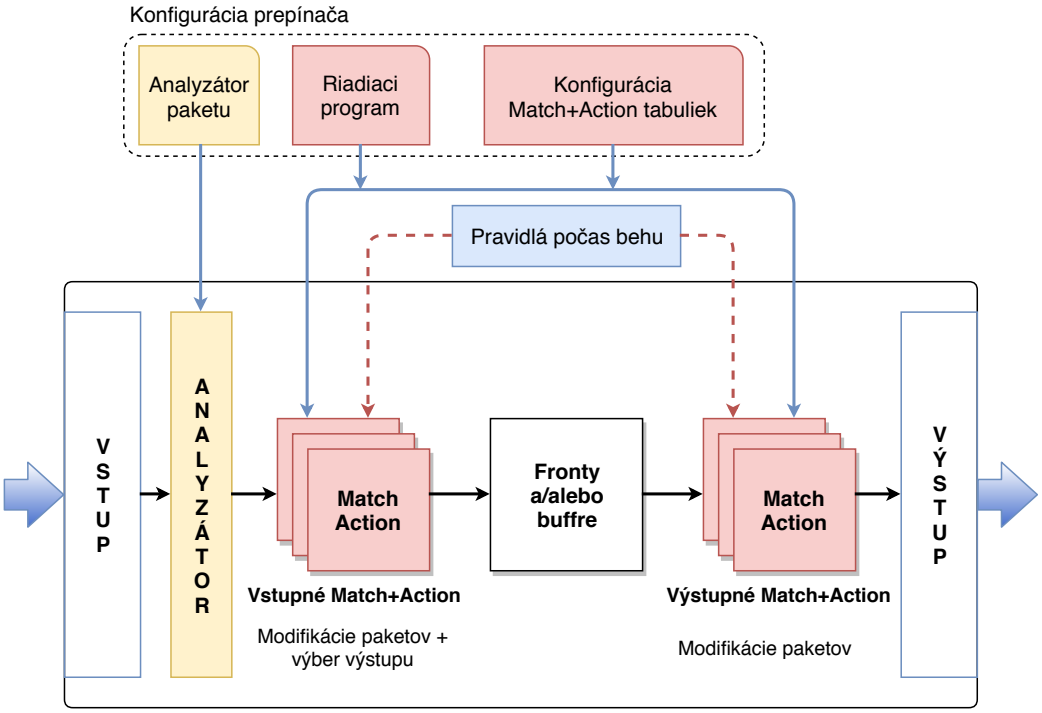
Definície tabuliek – definujú, ako sú porovnávané políčka protokolových hlavičiek so záznamami vo vyhľadávacích tabuľkách (napríklad presná zhoda, zhoda prefixu, vyhľadávanie podľa rozsahu), vstupné políčka použité pre hľadanie zhody, akcie, ktoré sa môžu aplikovať, a dimenzie tabuliek.

Definície akcií – popisujú pridružené akcie, ktoré sa majú vykonávať s paketmi daného toku. Jazyk P4 obsahuje určité primitívne akcie a umožňuje vytvárať aj užívateľsky definované akcie, ktoré sa skladajú z primitívnych alebo iných užívateľsky definovaných akcií.

[2] Z toho je odvodený názov – *match* = zhoda, *action* = akcia.

Špecifikácia riadiaceho programu – definuje usporiadanie tabuliek a postupnosť spracovania paketov.

Na obrázku 4.2 je zobrazená schéma abstraktného modelu, na ktorom je jazyk P4 založený. Pre každý vstupujúci paket vytvorí analyzátor rozloženú reprezentáciu (*parsed representation*) paketu, s ktorou narábajú *match+action* tabuľky. *Match+action* tabuľky vo vstupnej linke vytvoria výstupnú špecifikáciu určujúcu množinu portov, na ktoré bude paket poslaný. Spracovaním výstupnej špecifikácie sú vytvorené potrebné inštancie paketu a tie sú následne odovzdané výstupnej linke. Ak je to potrebné, pakety môžu byť pozdržané v buffroch. Po kompletnom spracovaní paketu vo výstupnej linke je z rozloženej reprezentácie paketu poskladaná hlavička a paket je odoslaný na výstup.



Obr. 4.2: Vysokoúrovňová schéma abstraktného modelu pre jazyk P4.

Jazyk P4 sa zameriava na špecifikáciu analyzátoru, *match+action* tabuliek a riadiaceho programu linky. Vytvorený program v jazyku P4 potom špecifikuje konfiguráciu prepínača. P4 program môže byť cieľovým zariadením priamo vykonávaný alebo môže byť preložený do vhodnej reprezentácie pre cieľové zariadenie.

V rámci projektu Liberouter v združení CESNET bol vytvorený prekladač z jazyka P4 do VHDL [1], ktorý umožňuje automatické generovanie aplikačného jadra firmvéru pre analýzu a spracovanie paketov v FPGA z jednoduchého popisu v P4. Vygenerovaný analyzátor paketov dosahuje priepustnosť až 100 Gb/s podobne ako optimalizované riešenie napísané v jazyku pre popis hardvéru (VHDL) bez nutnosti poznať tento jazyk. Vďaka tomu je možné zamerať sa priamo na popis funkcie, ktorú má sieťové zariadenie vykonávať, namiesto programovania FPGA. Prekladač podporuje verziu jazyka $P4_{14}$. Pre ovládanie aplikačného jadra generovaného z popisu v P4 existuje knižnica `libp4dev`. Hlavnou funkcionalitou, ktorú knižnica poskytuje je rozhranie pre zapnutie a zastavenie linky spracovania paketov, inicializáciu tabuliek a nahrávanie pravidiel do tabuliek.

Základnými prvkami jazyka P4, ktoré boli použité pri popise firmvéru v rámci tejto práce sú: typy hlavičiek, inštancie hlavičiek a metadát, kontrolné súčty a generátory hash hodnôt, špecifikácia analyzátoru, akcie, tabuľky, riadiaci program.

Typy hlavičiek popisujú usporiadanie položiek a definujú ich mená pre ďalšie referencovanie. Sú použité pri vytváraní inštancií hlavičiek paketov a metadát. Paket môže obsahovať niekoľko inštancií daného typu. Paketové hlavičky sú obvykle určené, keď sa paket objaví na vstupe. Metadáta väčšinou obsahujú informácie, ktoré sa nenachádzajú v dátach paketu a slúžia pre uchovávanie stavu počas spracovania daného paketu. Rozdiel medzi paketovými hlavičkami a metadátami je v tom, kedy sú považované za platné. Paketové hlavičky obsahujú špeciálny bit, ktorý určuje, či je hlavička platná, a môže byť explicitne testovaný. Metadáta sú vždy platné a inicializované sú hodnotou 0.

Kontrolné súčty a generátory hash hodnôt môžu byť definované ako funkcie, ktoré sú asociované s množinou políčok a výsledok je referencovateľný v rámci P4 programu. Základné definované algoritmy pre výpočet výsledku z množiny vstupných políčok sú *xor16*, *csum16*, *crc16*, *crc32* a *programmable_crc*.

Analyzátor je modelovaný ako stavový automat vytvárajúci rozloženú reprezentáciu paketu, s ktorou pracujú *match+action* tabuľky. Rozložená reprezentácia paketu je množina inštancií metadát a paketových hlavičiek, ktoré sú platné pre daný paket. *Match+action* tabuľky môžu meniť hodnoty políčok a platnosť inštancií hlavičiek, a tak upravovať rozloženú reprezentáciu paketu.

Akcie sú deklarované ako funkcie. Ich mená sa používajú pri upravovaní tabuľky počas behu k priradeniu ku záznamu v tabuľke. Takéto akcie sa nazývajú zložené. Okrem zložených akcií je v jazyku P4 definovaná množina primitívnych akcií. Zložené akcie sú poskladané z primitívnych akcií. Funkcie akcií môžu obsahovať parametre. Hodnoty parametrov sú naprogramované do záznamu v tabuľke počas behu. Keď nastane zhoda, parametre sú predané akcii.

Tabuľky deklarujú zoznam políčok, ktoré sú použité pre určovanie zhody paketu so záznamom v tabuľke, a akcie, ktoré je možné priradiť k záznamom v tabuľke. Pri hľadaní zhody sa pre daný paket musia zhodovať všetky špecifikované políčka záznamu v tabuľke. Pri vkladaní záznamu do tabuľky počas behu musia byť špecifikované hodnoty všetkých deklarovaných políčok. Ak nie je nájdená žiadna zhoda pre paket, vykonáva sa východzia akcia. Východzia akcia musí byť špecifikovaná počas behu.

Riadiaci program popisuje sekvenciu *match+action* tabuliek, ktorými je paket spracovaný. V riadiacom programe môžu byť aplikované tabuľky, volané iné riadiace funkcie alebo testované podmienky. Po aplikovaní tabuľky sú tri možnosti, ako sa bude ďalej pokračovať so spracovaním paketu: bezpodmienečné pokračovanie ďalším príkazom, pokračovanie blokom príkazov zvoleným podľa vykonanej akcie, alebo pokračovanie blokom zvoleným na základe toho, či nastala zhoda (*hit*) alebo nie (*miss*).

V roku 2016 bola vydaná aktualizovaná verzia jazyka P4, označovaná ako $P4_{16}$ [31], ktorá zavádza do jazyka P4 spätne nekompatibilné zmeny v porovnaní s verziou $P4_{14}$ [30]. Verzia $P4_{16}$ odstraňuje množstvo vlastností jazyka, ako napríklad čítače (*counters*) a jednotky pre kontrolné súčty, a presúva ich do knižníc. Jazyk bol tak zjednodušený z komplexného jazyka na podstatne menší jazyk, ktorý je doplnený knižnicou poskytujúcou základné konštrukcie potrebné pre väčšinu P4 programov.

Kapitola 5

DPDK pre karty COMBO

Pre sieťové karty COMBO bol v rámci bakalárskej práce [39] vytvorený ovládač do DPDK, ktorý zabezpečuje prenosy do DPDK cez rozhranie SZE2. Tento ovládač využíva knižnicu *libsze2* a bol nazvaný **szedata2** podľa názvu modulu jadra pre rozhranie SZE2. Ovládač szedata2 je súčasťou hlavnej vývojovej vetvy DPDK od verzie 2.2.0 (december 2015). V ovládači sú implementované funkcie pre prijímanie paketov (`rx_pkt_burst()`), odosielanie paketov (`tx_pkt_burst()`) a základné funkcie pre konfiguráciu karty.

Prijímanie a odosielanie dát v DPDK funguje na princípe deskriptorov. V RX smere napĺňa ovládač deskriptory fyzickými adresami pripravených mbufov a odosiela ich zariadeniu. Zariadenie odosiela pakety DMA prenosmi priamo do mbufov a ovládač potom posúva mbufy s prijatými paketmi aplikácii. V TX smere ovládač obdrží od aplikácie mbufy s paketmi na odoslanie, uloží adresy mbufov do deskriptorov a odosiela deskriptory zariadeniu. Zariadenie pakety odošle do siete a oznámi ovládaču, že deskriptor je voľný. Ovládač potom môže uvoľniť mbuf odkazovaný deskriptorom a znovu použiť deskriptor.

Model prijímania a odosielania dát v DPDK ovládači szedata2 sa oproti klasickému prístupu v DPDK líši. Rozhranie SZE2 nefunguje na princípe deskriptorov. Namiesto toho sú dáta ukladané do súvislých kruhových buffrov. Pakety je preto potrebné dostať z kruhového buffru do mbufov v RX smere a z mbufov do kruhového buffru v TX smere. V RX smere ovládač prostredníctvom API knižnice libsze2 zamyká kruhový buffer, nakopíruje pakety do mbufov, odomkne časť kruhového buffru, z ktorej boli prevzaté pakety a posunie pakety aplikácii. V TX smere si ovládač vypýta prostredníctvom API knižnice libsze2 voľné miesto v kruhovom buffri, uloží do neho pakety na odoslanie a oznámi zariadeniu, že je pakety možné odoslať.

Kvôli potrebe kopírovať paketové dáta sa nadmerne zaťažuje CPU a pamäťový subsystém. Preto boli navrhnuté DMA prenosy, ktoré prenášajú dáta spôsobom prirodzeným pre DPDK – prostredníctvom deskriptorov. Paketové dáta sú tak DMA prenosmi prenášané priamo medzi mbufmi a bufframi v karte. Vytvorený bol ovládač pre DMA prenosy pracujúce na princípe deskriptorov, ktorý sa od ovládača szedata2 líši v spôsobe inicializácie a riadenia dátových prenosov. Ostatné konfiguračné funkcie ostávajú rovnaké.

Deskriptory majú v RX aj TX smere veľkosť 64 bitov a sú uložené v kruhovom buffri. Zaplnenie buffru sa určuje podobne ako pri SZE2 buffri ukazovateľmi na hranice obsadených a voľných deskriptorov. Počet deskriptorov je nastaviteľný pri konfigurácii fronty zariadenia. Podporované sú iba počty deskriptorov, ktoré sú mocninou čísla 2. Ovládač má za úlohu tieto buffre alokovať pri inicializácii front a uložiť ich fyzické adresy do príslušných registrov.

V RX smere nesie deskriptor fyzickú adresu buffru v mbufe, do ktorého je možné preniesť paket a veľkosť buffru. Ak by bol paket väčší, je potrebné ho preniesť vo viacerých

deskriptoroch a mbufy zreťaziť. 48 bitov deskriptoru je použitých pre fyzickú adresu a horných 16 bitov pre veľkosť buffru. Horných 16 bitov fyzickej adresy mbufu musí byť zhodných s adresou buffru pre deskriptory. Spolu s paketom je prenášaná aj hlavička, z ktorej ovládač vytiahne dĺžku paketu, rovnako ako u rozhrania SZE2.

V TX smere obsahuje deskriptor podobne ako pri RX smere 48 bitov adresy mbufu. V prípade, že by sa horných 16 bitov nezhodovalo s adresou buffru pre deskriptory, nie je možné takýto paket preniesť. Avšak v súčasnosti je 48 bitov pre fyzickú adresu postačujúcich. 15 bitov je použitých pre celkovú veľkosť paketu v buffri a 1 najvrchnejší bit určuje, či sú mbufy zreťazené alebo nie.

5.1 Merania výkonnosti

V tejto sekcii sú popísané merania výkonnosti a výsledky dosiahnuté s DPDK ovládačom szedata2 a ovládačom pre deskriptorové prenosy. Odmeraná bola rýchlosť prenosu paketov pri prijímaní (RX) a odosielaní (TX) pre rôzne počty použitých DMA front (jadier CPU) s testovacou DPDK aplikáciou `testpmd` [47], ktorá slúži na testovanie DPDK ovládačov pre sieťové zariadenia a umožňuje prijímanie, odosielanie a jednoduché preposielanie paketov.

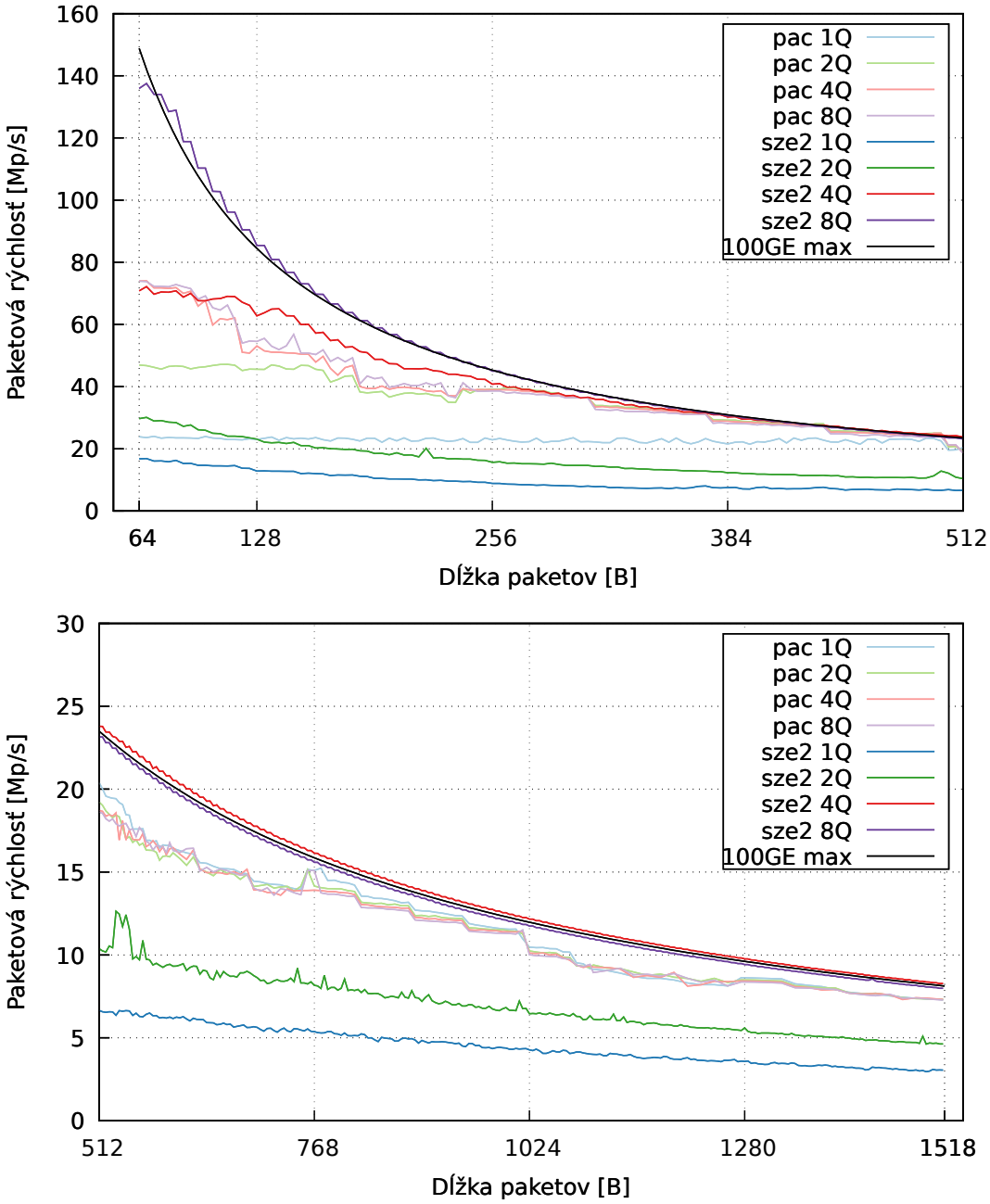
Merania boli vykonané na stroji s kartou COMBO-100G1 s firmvérom obsahujúcim interný generátor dát priamo v FPGA namiesto ethernetového rozhrania a doplnkovými čítačmi štatistík pre DMA prenosy. Takto bolo možné naplno zaťažiť zbernicu PCI-Express aj nad maximálnu priepustnosť 100G Ethernetu. V tabuľke 5.1 sú zhrnuté informácie o použitom stroji a softvéri.

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

Informácie o použitom stroji (Dell PowerEdge R730):	
Základná doska	Dell Inc. 0H21J3
CPU	Xeon(R) CPU E5-2660 v3 @ 2,60 GHz
Počet jadier	2x10 (2x20)
Hyper-Threading	áno
RAM	64GB DDR4 @ 2133 MHz
Informácie o OS a použitom softvéri:	
OS	Scientific Linux verzia 6.5 (Carbon)
Kernel	2.6.32-431.1.2.el6.x86_64
Verzia DPDK	16.11
DPDK aplikácia	`testpmd`

Kedže pre získanie podrobného profilu naprieč všetkými paketovými dĺžkami bolo potrebné vykonať veľké množstvo meraní, meranie pre jednu paketovú dĺžku trvalo 1 sekundu. Paketové dĺžky v grafoch sú počítané vrátane kontrolného súčtu v ethernetovom rámci. Z čítača počtu hodinových signálov a známej frekvencie firmvéru bola vypočítaná presná doba behu daného merania. Z presnej doby merania a počtu prenesených paketov bola vypočítaná paketová rýchlosť.

Na obrázku 5.1 sú zobrazené grafy porovnávajúce rýchlosti preposielania paketov (t. j. prijatie a následne odoslanie paketov) s ovládačom szedata2 a s ovládačom pre deskriptorové prenosy pre 1, 2, 4 a 8 jadier CPU. Pre 1 a 2 jadrá CPU je dosahovaná s ovládačom pre deskriptorové prenosy výrazne vyššia paketová rýchlosť, ale rýchlosť linky nie je možné



Obr. 5.1: Paketová rýchlosť pri preposielaní dát (pac – deskriptorové DMA prenosy, sze2 – ovládač szedata2, Q – počet použitých DMA front/jadier CPU).

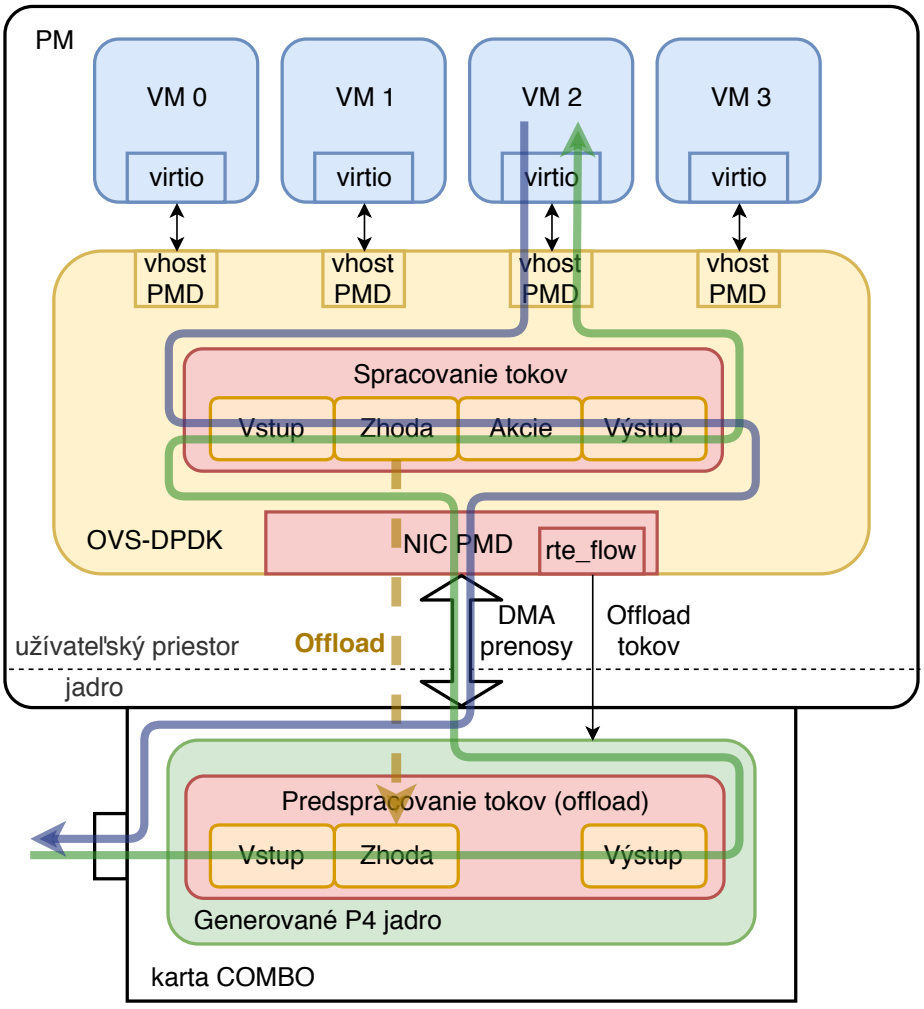
dosiahnuť ani s vyšším počtom jadier CPU. Limitujúcim faktorom sú DMA prenosy s deskriptormi, ktoré využívajú prenosové pásmo menej efektívne ako rozhranie SZE2. S ovládačom szedata2 je naproti tomu možné s dostatočným počtom jadier CPU dosiahnuť aj plnú rýchlosť linky.

Kapitola 6

Akcelerácia OVS s kartami COMBO

V sekcii 3.2 je popísaný spôsob, akým bol použitý framework DPDK k zlepšeniu výkonnosti OVS. Použitím OVS-DPDK bola so sieťovými kartami od Intelu dosahovaná 10 – 18-krát vyššia priepustnosť s topológiou P2P a 3 – 7-krát vyššia priepustnosť s topológiou PVP. Súčasťou DPDK je aj ovládač szedata2 pre sieťové karty COMBO, ktorého výkonnosť bola popísaná v kapitole 5. Ovládač szedata2 riadi prenosy dát cez rýchle DMA rozhranie SZE2, ktoré funguje na princípe prúdového prenosu dát. Okrem prenosov rozhraním SZE2 je v projekte Liberouter vyvíjaný aj nový typ DMA prenosov využívajúcich deskriptory spôsobom prirodzeným pre DPDK. Pre nový typ DMA prenosov bol vytvorený experimentálny DPDK ovládač, ktorého výkonnosť bola porovnaná s ovládačom szedata2. Výhodou tohto ovládača je, že nie je nutné kopírovať dáta paketov do buffrov v DPDK tak, ako sa to deje v prípade ovládača szedata2. Preto je možné dosahovať vyššiu rýchlosť prenosu paketov s nižším počtom jadier, avšak nie je možné dosiahnuť maximálnu rýchlosť 100G linky. Firmvér pre deskriptorové DMA prenosy je však v súčasnosti experimentálny a pre kartu COMBO-200G2QL nie je plne funkčný. Preto bol pre túto prácu zvolený typ DMA prenosov cez rozhranie SZE2 a DPDK ovládač szedata2. V rámci práce bola do ovládača szedata2 pridaná podpora nových kariet COMBO-200G2QL. Prvým krokom k akcelerácii OVS s kartami COMBO je použitie DPDK.

V kapitole 3 sú ukázané viaceré prístupy k akcelerácii OVS od rôznych výrobcov sieťových zariadení. Použitím offloadu spracovania tokov do sieťového zariadenia je možné zvýšiť priepustnosť celého virtualizovaného systému a zároveň znížiť zaťaženie CPU vykonávaním činnosti OVS a ušetriť tak prostriedky pre virtuálne stroje. API pre offload pravidiel do hardvéru je súčasťou OVS od júna 2017. Pre toto API je implementovaná podpora štandardného linuxového rozhrania s dátovou cestou v jadre. Pri tomto prístupe sa pre doručenie paketov do cieľovej virtuálnej stanice typicky používajú SR-IOV virtuálne funkcie. Komunita vývojárov v projektoch OVS a DPDK pracuje na pridaní podpory offloadu do dátovej cesty implementovanej v užívateľskom priestore cez DPDK. Offload do hardvéru cez DPDK je možný dvomi prístupmi: čiastočný offload a plný offload [9]. Plný offload do hardvéru cez DPDK v súčasnosti nie je podporovaný a vyžaduje pridanie knižníc do DPDK, na ktorých pracuje najmä spoločnosť Intel. Pre čiastočný offload cez DPDK bola vytvorená sada patchov pre OVS od spoločností Napatech a Mellanox Technologies, pre ktoré je plánované pridanie do nasledujúceho vydania OVS 2.10.0. Tieto patche využívajú DPDK API rte_flow k nahrávaniu pravidiel do sieťovej karty.



Obr. 6.1: Schéma princípu čiastočného offloadu tokov z OVS do hardvéru. Klasifikácia paketov do tokov (hľadanie zhody v tabuľkách tokov), može byť presunutá do sieťovej karty prostredníctvom API rte_flow. Offloadovaná časť z linky pre spracovanie paketu je znázornená tmavožltou prerušovanou šípkou. Priechod paketov systémom až do virtuálneho stroja je znázornený zelenou šípkou. Priechod paketov z virtuálneho stroja do fyzickej siete je znázornený fialovou šípkou.

Druhým krokom k akcelerácii OVS v tejto práci je použitie čiastočného offloadu tokov do hardvéru cez DPDK. Na obrázku 6.1 je schéma zobrazujúca princíp čiastočného offloadu. Jadro pre spracovanie paketov vo firmvéri karty s FPGA čipom je generované z popisu v jazyku P4 a umožňuje predspracovanie prichádzajúcich paketov. Pakety môžu byť klasifikované do príslušných tokov, a poslané cez DMA prenosy do ďalšieho spracovania dátovej cesty OVS v užívateľskom priestore. OVS obdrží pakety cez ovládač sieťového zariadenia (PMD) v DPDK a záznamy z tabuliek tokov môže presunúť do sieťovej karty (*offload*) s využitím DPDK API rte_flow. Ak je prichádzajúci paket pri predspracovaní v karte nájdený v jej tabuľkách tokov, dostane špeciálnu značku identifikujúcu záznam v tabuľke tokov a v dátovej ceste v softvéri môže byť preskočená klasifikácia takéhoto paketu pričom sú priamo aplikované akcie zodpovedajúceho záznamu z tabuľky tokov a paket môže byť doručený do správneho virtuálneho stroja. Ak prichádzajúci paket nemôže byť klasifikovaný

v sieťovej karte, musí prejsť cez celú dátovú cestu v softvéri. Príslušné záznamy tokov z tabuliek môžu byť presunuté do karty a ďalšie pakety z rovnakého toku sú potom klasifikované pri spracovaní v FPGA.

V sekcii 6.1 je popísaný princíp patchov od spoločností Napatech a Mellanox Technologies pridávajúcich do OVS podporu čiastočného offloadu cez API rte_flow. Aby bolo možné využiť čiastočný offload tokov z OVS, musí byť v karte COMBO použitý príslušný firmvér, ktorý takýto offload podporuje. Popis firmvéru nasleduje v sekcii 6.2. Do DPDK ovládača szedata2 bola pridaná podpora API rte_flow. Popis podpory rte_flow v ovládači szedata2 je v sekcii 6.3.

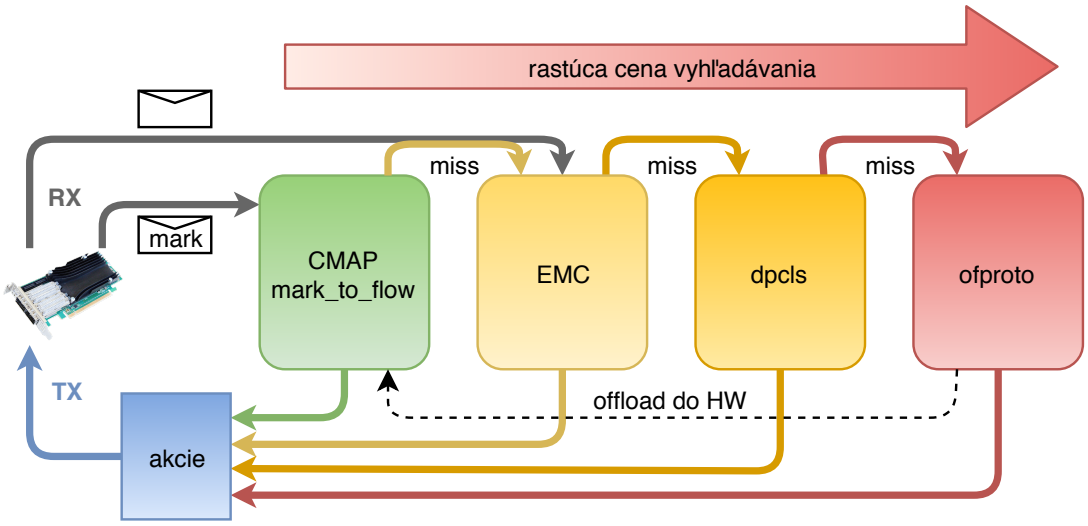
6.1 Čiastočný offload tokov z OVS cez API rte_flow

Čiastočný offload využíva priradenie značiek toku paketov. Každý paket z daného toku dostane v hardvéri značku, ktorá potom v dátovej ceste umožňuje získať pravidlo pre daný tok priamo na základe jej hodnoty, čo ušetrí extrahovanie miniflow, vyhľadávanie v EMC, prípadne v *dpcls* (popísané v sekcii 3.2.2). Značka je bezznamienkové celé číslo s veľkosťou 4 B (`uint32_t`). Priradenie značiek (*mark id*) k tokom je v dátovej ceste implementované pomocou kukučej hash tabuľky (CMAP) umožňujúcej súbežný prístup jednému zapisujúcemu a ľubovoľnému počtu čítajúcich vlákien bez zamykania. Do hardvéru je značka nahraná pomocou rte_flow akcie MARK.

Každé PMD vlákno má vlastnú tabuľku tokov. Pre jeden mega tok (*mega flow*) tak môžu existovať záznamy (`dp_netdev_flow`) v tabuľkách tokov viacerých vlákien, čo by znamenalo, že by rovnaké pravidlo mohlo byť nahrávané do hardvéru viackrát s rôznou značkou. Preto je vytvorená CMAP tabuľka `megaflow_to_mark`, ktorá priraďuje mega toky (*mega flow*) k značkám v pomere 1:1. Záznam do tabuľky vytvára len prvé vlákno, ktoré chce offloadovať mega tok do hardvéru. Každé ďalšie vlákno sa z tabuľky dozvie, že príslušné pravidlo už je nahrané v hardvéri a nebude sa ho pokúšať nahrávať znovu. Opačné mapovanie zo značky na tok (`dp_netdev_flow`) je v pomere 1:N, čo zabezpečuje CMAP tabuľka `mark_to_flow`. Prvé PMD vlákno, ktoré chce offloadovať pravidlo pre mega tok, si nechá prideliť nové číslo značky a zavolá funkciu z API pre offload pravidla. Ak je úspešné, do tabuľky sa pridá nový záznam priraďujúci tok (`dp_netdev_flow`) značke. Ďalšie PMD vlákna získajú príslušné číslo značky z tabuľky `megaflow_to_mark` a potom vytvoria ďalší záznam v tabuľke `mark_to_flow`.

Keď PMD vlákno obsluhuje prijatý paket, ešte pred vyhľadávaním v EMC overí či paket neobsahuje značku. Značka je uložená v štruktúre *mbufu* v políčku `hash.fdir.hi` a jej prítomnosť je označená flagom `PKT_RX_FDIR_ID` v políčku `of_flags`. Ak paket obsahuje značku, príslušný tok je určený vyhľadaním v tabuľke `mark_to_flow`. Inak sa pokračuje klasickým vyhľadávaním v EMC, *dpcls*, prípadne OpenFlow tabuľkách. Tento proces je naznačený na obrázku 6.2.

Podnet pre offload do hardvéru vzniká, keď pre paket nie je nájdený tok v EMC, ani v *dpcls*, a musí byť vyhľadávaný až v OpenFlow tabuľkách (tzv. *upcall*), čo je súčasťou dátovej cesty. Keďže operácia vkladania a modifikovania pravidla pre tok do hardvéru je časovo náročná, nie je vhodné, aby bola vykonávaná vláknom obsluhujúcim dátovú cestu, pretože pri častom vkladaní alebo modifikovaní pravidiel by dochádzalo k zahlcovaniu dátovej cesty a stratám paketov. Preto sú všetky operácie s pravidlami tokov ukladané do zoznamu obsluhovaného ďalším vláknom, ktoré vykonáva skutočné volanie operácií vytvárania a mazania pravidiel cez API rte_flow. Modifikácia pravidla prebieha tak, že je pravidlo zmazané a potom znovu vytvorené.



Obr. 6.2: Postup určovania toku pre prijatý paket. Ak paket obsahuje značku (*mark*), tak je vyhľadaný v CMAP tabuľke `mark_to_flow`. Ak značku neobsahuje alebo nie je nájdený pokračuje postupne do tabuliek EMC, *dpcls* alebo *ofproto* (implementácia OpenFlow tabuliek), do kým mu nie je priradený príslušný tok. S každou ďalšou tabuľkou rastie cena vyhľadania paketu. Offload do hardvéru je vyžiadaný pri vyhľadávaní toku v OpenFlow tabuľkách.

Pravidlo je vytvorené volaním funkcie `rte_flow_create()`. Pred zavolaním je potrebné pripraviť príslušné parametre: atribúty, vzor a akcie. Atribúty nie sú používané. Vzor určuje špecifikáciu hodnôt políčok hlavičiek, prípadne masky pomocou poľa položiek (`struct rte_flow_item`). Prehľad podporovaných položiek je v tabuľke 6.1. Na záver sú pravidlu pridané akcie `RSS`, `MARK` a `END`. Akcia `RSS` je pridaná, pretože samotná akcia `MARK` nie je podporovaná rozhraním rte_flow a musí byť použitá v kombinácii s ukončujúcou akciou, čo je napríklad `RSS`. Akcia `RSS` má nastavenú konfiguračnú štruktúru na `NULL`, čo znamená, že je použité východzie RSS nastavenie. Akcia `MARK` obsahuje v konfiguračnej štruktúre jeden prvok `id`, ktorý je nastavený na hodnotu značky, ktorá sa má prideľovať paketom daného toku. Na koniec je pridaná akcia `END`, ktorá značí koniec poľa akcií.

6.2 Firmvér pre podporu offloadu cez API rte_flow

K vytvoreniu firmvéru podporujúceho offload pravidiel z OVS do hardvéru bol použitý popis v jazyku P4 a prekladač z P4 do VHDL, ktorý bol spomenutý v skecii 4.3. Na obrázku 6.3 je znázornená vysokoúrovňová schéma firmvéru pre kartu COMBO-100G2Q[1]. Modré šípky znázorňujú cestu prijatého paketu (RX), červená šípka priechod linkou generovanou z P4, zelená šípka cestu paketu posielaného do siete (TX). Firmvér pre všeobecný P4 program má do linky pre spracovanie paketov zapojenú aj cestu v TX smere, čo je znázornené šedou prerušovanou čiarou. Vo firmvéri pre OVS nie je v TX smere potrebné žiadne spracovanie paketov. Preto bol firmvér upravený tak, aby pakety v TX smere neprechádzali linkou generovanou z P4, ale išli priamo do sieťového modulu. Inak by museli pakety v RX aj TX smere zdieľať linku generovanú z P4, čo by znamenalo nižšiu výslednú priepustnosť. Kompletný firmvér nie je dielom tejto práce. V rámci práce bol vytvorený popis v jazyku P4,

[1]Firmvér pre kartu COMBO-200G2QL obsahuje zobrazené moduly dvakrát.

Tabuľka 6.1: Prehľad podporovaných položiek vzoru a ich políčok pre rte_flow pravidlá.

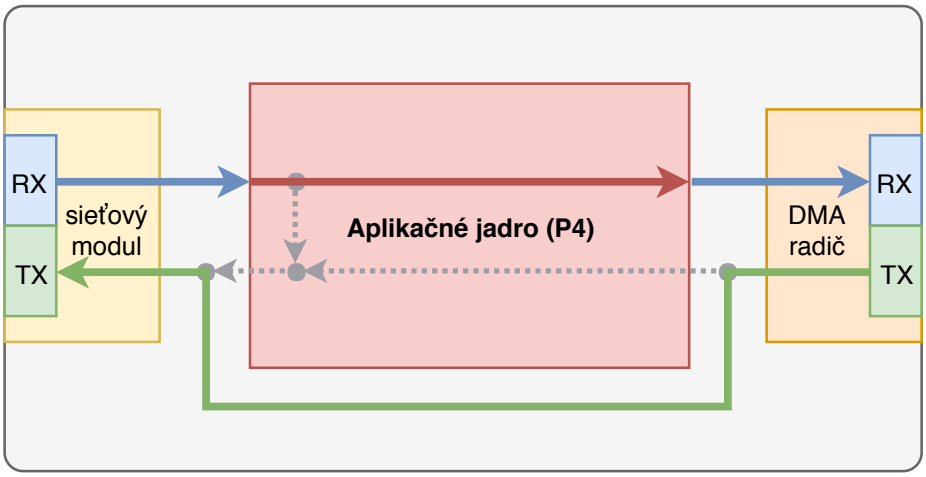
Protokol	Typ položky	Podporované políčka
Ethernet	RTE_FLOW_ITEM_TYPE_ETH	dst, src, type
VLAN	RTE_FLOW_ITEM_TYPE_VLAN	tci
IPv4	RTE_FLOW_ITEM_TYPE_IPV4	type_of_service, time_to_live, next_proto_id, src_addr, dst_addr
ICMP	RTE_FLOW_ITEM_TYPE_ICMP	icmp_type, icmp_code
TCP	RTE_FLOW_ITEM_TYPE_TCP	src_port, dst_port, data_off, tcp_flags
UDP	RTE_FLOW_ITEM_TYPE_UDP	src_port, dst_port
SCTP	RTE_FLOW_ITEM_TYPE_SCTP	src_port, dst_port

z ktorého je generované aplikačné jadro firmvéru. V čase písania tejto práce nebol hotový firmvér pre kartu COMBO-200G2QL, preto bola pri testovaní použitá karta COMBO-100G2Q.

V P4 popise aplikačného jadra je definovaných všetkých sedem hlavičiek protokolov, pre ktoré je podporovaný offload do hardvéru z OVS cez API rte_flow (tabuľka 6.1). Okrem hlavičiek protokolov sú definované ešte dve hlavičky pre metadáta. Jedna hlavička (`match_fields_t`) obsahuje všetky políčka podporovaných protokolov, pre ktoré je podporovaný offload cez API rte_flow, a jeden bit pre každú hlavičku protokolu určujúci, či je hlavička prítomná alebo nie. Druhá hlavička (`fields_hash_t`) obsahuje políčko pre výpočet hash hodnoty, ktorá sa používa pre určenie RX DMA kanálu. Z každej hlavičky je vytvorená jedna inštancia. Okrem hlavičiek definovaných a inštancovaných v P4 programe zahŕňa prekladač z P4 do VHDL aj súbor, kde je definovaná a inštancovaná hlavička s vnútornými metadátami (`intrinsic_metadata_t`). Tieto sú spoločné pre všetky firmvéry generované z P4 a sú napojené na signály vstupujúce alebo vystupujúce z generovanej linky. Patrí sem napríklad číslo vstupného port (`ingress_port`) a výstupného portu (`egress_port`), pričom portom sa rozumie sieťové rozhranie alebo DMA kanál. Čísla sieťových rozhraní sú v rozsahu 0 – 127, čísla DMA kanálov 128 – 255. Do vnútorných metadát bola pridaná hodnota značky `mark_id` s veľkosťou 32 bitov. Hodnota značky je po výstupe z generovanej linky vložená do hardvérových metadát v SZE2 hlavičke, hneď za veľkosť celého SZE2 rámca a veľkosť hardvérových metadát, t. j. na offset 4 od začiatku SZE2 rámca. Žiadne ďalšie položky z vnútorných metadát nie sú do SZE2 hlavičky vkladané, takže celková veľkosť SZE2 hlavičky je 8 B.

Schéma na obrázku 6.4 zobrazuje postupnosť *match+action* tabuliek v P4 popise. Po vstupe do linky je paket prevedený do rozloženej reprezentácie. Následne je aplikovaná tabuľka pre každý z podporovaných protokolov nachádzajúcich sa v pakete (`eth`, `vlan`, `ipv4`, `icmp`, `tcp`, `udp`, `sctp`), ktorá uloží hodnotu políčok daného protokolu do metadát a nastaví políčko značiace prítomnosť hlavičky.

Potom je aplikovaná tabuľka `mark_id`, do ktorej sú počas behu nahrané pravidlá pre hľadanie zhody paketov. Každé pravidlo obsahuje hodnoty všetkých porovnávaných políčok, masky, akciu a parameter akcie. Akcia vždy nastavuje hodnotu `mark_id` podľa hodnoty parametru. Východzie pravidlo, ktoré sa aplikuje, keď nie je nájdená zhoda pre paket, prideľuje hodnotu `0xFFFFFFFF`, čo je hodnota značky, ktorá je považovaná za nevalidnú. Ak je nastavená táto hodnota, ovládač v DPDK považuje paket za neoznačkovaný. Aby

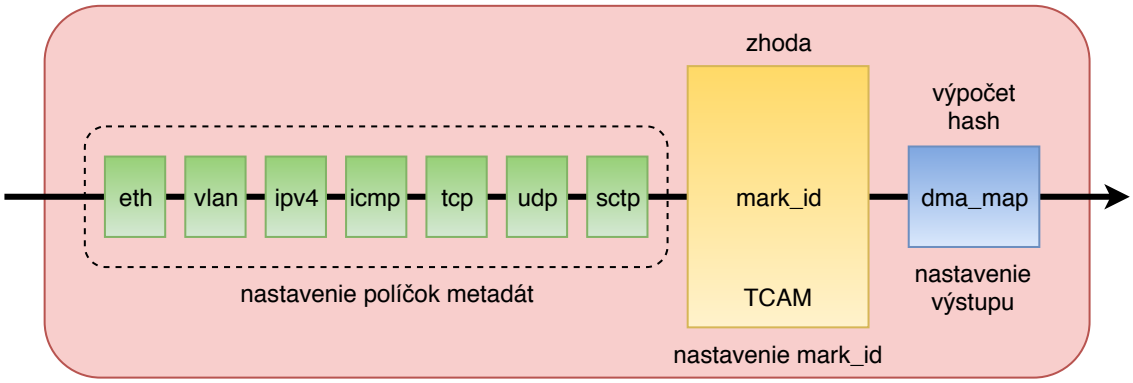


Obr. 6.3: Vysokoúrovňová schéma firmvéru pre kartu COMBO-100G2Q. Modré šípky znázorňujú cestu prijatého paketu (RX), červená šípka priechod linkou generovanou z P4, zelená šípka cestu paketu posielaného do siete (TX). Šedá prerušovaná šípka znázorňuje cestu paketu pre všeobecný P4 program. Vo výslednom firmvéri bola táto cesta odstránená, kvôli dosiahnutiu vyššej priepustnosti pre RX smer, pretože priepustnosť linky je inak zdieľaná pre oba smery. Keďže pakety v TX smere nevyžadujú žiadne spracovanie, mohla byť vytvorená alternatívna cesta obchádzajúca linku generovanú z P4.

tabuľka `mark_id` mohla podporovať aj použitie masiek pre pravidlá, typ zhody pre všetky položky je nastavený ako `ternary`, a teda tabuľka je implementovaná ako TCAM[2] tabuľka. Dôležitý parameter vytvoreného firmvéru je maximálny podporovaný počet riadkov tabuľky (podporovaných pravidiel). Vo firmvéri pre kartu COMBO-100G2Q bola nastavená veľkosť tabuľky na 31 riadkov, čo je pre reálne použitie nízka hodnota. Aj s takouto veľkosťou bol ale pri preklade firmvéru problém so splnením časovania, pričom len jeden z mnohých prekladov splnil časovanie. Karta COMBO-200G2QL obsahuje FPGA s väčším počtom zdrojov, takže sa dá predpokladať, že bude možné pripraviť firmvér s väčším počtom riadkov TCAM tabuľky.

Na záver je aplikovaná tabuľka `dma_map`, ktorá nastavuje hodnotu výstupného portu `egress_port` vo vnútorných metadátach, a tak posiela paket na určitý DMA kanál. V tabuľke je najprv vypočítaná hash hodnota, ktorá sa počíta zo všetkých položiek, nad ktorými sa hľadá zhoda v tabuľke `mark_id`. Použitá je funkcia CRC32. Maximálna hodnota je určená počtom DMA kanálov. Hash hodnota je potom pripočítaná k číslu 128, čím sa určí DMA kanál, do ktorého bude paket poslaný. V súčasnej implementácii je výsledný kanál určovaný zo všetkých dostupných DMA kanálov a nie je možné obmedziť maximálne číslo kanálu, pretože v prekladači z P4 do VHDL je chyba v implementácii výpočtu hash funkcie, ktorá zabraňuje tomu, aby bolo možné parametrizovať maximálnu hodnotu vypočítanej hash hodnoty parametrom akcie, čo by umožňovalo obmedziť maximálny počet kanálov počas behu nahraním pravidla do tabuľky.

[2] *Ternary Content-addressable Memory*



Obr. 6.4: Schéma linky pre spracovanie paketov generovanej z popisu v jazyku P4. Linka obsahuje tabuľky, ktoré nastavujú hodnoty políčok prítomných protokolov do metadát použitých k nájdeniu zhody a nastaveniu značky. Na konci linky je použitím hash hodnoty vypočítanej z políčok hlavičiek vybraný RX DMA kanál, do ktorého je paket poslaný.

6.3 Podpora API rte_flow v DPDK ovládači szedata2

Pre podporu API rte_flow musí ovládač implementovať typ filtra `RTE_ETH_FILTER_GENERIC` a operáciu `RTE_ETH_FILTER_GET`, ktorá volajúcemu vráti ukazovateľ na štruktúru obsahujúcu operácie filtra. Podpora filtra sa implementuje nastavením ukazovateľa na funkciu `filter_ctrl` v štruktúre `eth_dev_ops` obsahujúcej ukazovatele na funkcie, ktoré implementujú rozhranie ethernetových zariadení (*Ethernet Device API*). Do ovládača szedata2 bola pridaná funkcia `szedata2_filter_ctrl()`, ktorá implementuje uvedený typ filtra a operáciu. Žiadne iné typy filtra ani operácie nie sú podporované. K práci s pravidlami v hardvéri je použitá knižnica *libp4dev*.

V štruktúre `rte_flow_ops` sú nastavené funkcie API rte_flow, ktoré implementuje ovládač. Offload do hardvéru z OVS využíva len dve funkcie pre vytváranie pravidiel (`create`) a rušenie pravidiel (`destroy`). Tieto dve sú implementované v ovládači szedata2 a popisu ich implementácie sa venuje táto sekcia. Ďalšie funkcie pre podporu offloadu nie sú potrebné a preto neboli implementované. Okrem vytvárania a rušenia pravidiel je dôležitou súčasťou aj inicializácia linky spracúvajúcej pakety.

Kvôli podpore akcie `MARK`, ktorá označuje paket značkou, musela byť upravená aj dátová cesta v RX smere. Pri prijatí paketu je zo SZE2 hlavičky získaná hodnota značky. Za neplatnú hodnotu značky sa považuje hodnota `0xFFFFFFFF`. Ak má značka túto hodnotu, považuje sa za nenastavenú a pokračuje sa štandardným spracovaním paketu. Ak má značka platnú hodnotu, tak je v *mbufe* do položky `hash.fdir.hi` nastavená hodnota značky a v položke `ol_flags` je nastavený flag `PKT_RX_FDIR_ID`.

Aby správne fungovalo zastavovanie linky spracúvajúcej pakety, musí byť modul jadra `szedata2_cv3`, resp. `szedata2_cv3_fdt` načítaný s parametrom `discard_on_full=1`. Táto hodnota zabezpečí, že ak je DMA kanál plný, dáta na vstupe daného kanálu budú zahadzované. Ak by nebolo zapnuté zahadzovanie na vstupe do DMA kanálu pri jeho zaplnení, mohla by sa aplikácia zaseknúť v stave, kedy by sa čakalo na zastavenie linky, ktoré by nikdy nenastalo v prípade, že by bol naštartovaný DMA kanál, z ktorého by aplikácia neodoberala pakety. Ak by sa v takomto prípade aplikácia pokúsila zastaviť linku pre spracovanie paketov, k zastaveniu linky by nikdy nedošlo, pretože pri zastavovaní sa čaká na stav, kedy sú prázdne vnútorné buffre, ale tie by sa nevyprázdnili, do kým by neboli prečítané dáta

z daného DMA kanálu. Keď sa budú zahadzovať dáta na vstupe do DMA kanálu, ak je plný, tak sa vnútorné buffre v linke vyprázdnia a linka sa úspešne zastaví, aj keď z daného DMA kanálu pakety nie sú odoberané.

Inicializácia linky pre spracovanie paketov Pri inicializácii zariadenia v DPDK musí byť inicializovaná linka a všetky jej tabuľky. Inicializácia linky je implementovaná vo funkcii `p4_device_init()`. Štruktúry a informácie potrebné k obsluhe knižnice *libp4dev* sú uložené v štruktúre `p4_device` obsahujúcej: ukazovateľ na štruktúru `p4dev_t`, ktorá udržuje kontext a pracujú s ňou funkcie z knižnice *libp4dev*, zámok `rte_spinlock_t`, ktorý slúži na zabránenie súbežného prístupu k API knižnice *libp4dev*, a pole štruktúr `p4_table` s informáciami o *match+action* tabuľkách. Informácie o tabuľkách obsahujú: meno tabuľky, typ tabuľky, meno východzej akcie, parameter pre východziu akciu, aktuálny počet nahraných pravidiel, maximálny počet pravidiel a zoznam pravidiel. Zoznam pravidiel je uložený v abstraktnej štruktúre *tail queue*.

Pri inicializácii linky je najprv získaná cesta k špeciálnemu znakovému zariadeniu, ktorá sa zadáva ako parameter inicializačnej funkcii z knižnice *libp4dev* `p4dev_init()`. Následne ja zastavená linka pre prípad, že bola spustená pred zapnutím DPDK. Potom je zavolaná funkcia `p4dev_reset_device()`, ktorá v hardvéri inicializuje všetky *match+action* tabuľky. Po jej zavolaní sú všetky tabuľky prázdne a je potrebné nastaviť v nich východzie pravidlá. Nastavenie východzích pravidiel a uloženie informácií do internej štruktúry pre všetky tabuľky je implementované vo funkcii `p4_tables_init()`. Pravidlo sa označuje ako východzie volaním funkcie `p4rule_mark_default()`. Tabuľky sú inicializované na základe východzích hodnôt, ktoré sú uložené v poli `p4_tables`. Keď prebehne inicializácia tabuliek v poriadku, zavolá sa funkcia na spustenie linky a linka je pripravená na použitie.

Pri deinicializácii zariadenia v DPDK je taktiež deinicializovaná linka spracovania paketov, čo znamená, že linka je zastavená a všetky tabuľky sú inicializované do východzieho stavu.

Vytváranie pravidiel Všetky pravidlá tokov sú ukladané do tabuľky `table_mark_id`. Každé pravidlo pre P4 linku je asociované s tokom. Toky sú reprezentované štruktúrou `rte_flow`. Túto štruktúru si definuje každý ovládač a môže obsahovať položky, ktoré potrebuje daný ovládač. V ovládači szedata2 má každý tok uložený ukazovateľ na záznam v zozname pravidiel pre tabuľku `table_mark_id`. Zoznam pravidiel musí byť udržovaný preto, lebo knižnica *libp4dev* neumožňuje pridávať pravidlá po jednom, ale vždy musia byť pridávané všetky pravidlá naraz. Preto pri každom vytváraní alebo rušení pravidla musia byť nanovo nahrané všetky existujúce pravidlá. Zoznam všetkých vytvorených tokov je uložený v súkromnej štruktúre zariadenia v abstraktnej štruktúre *tail queue*. Vytváranie nového toku zabezpečuje funkcia `szedata2_flow_create()`.

Pri vytváraní nového toku prebehnú nasledujúce kroky: zamkne sa zámok v štruktúre `p4_device`; overí sa, či je v tabuľke `table_mark_id` voľné miesto pre nové pravidlo; skontroluje sa, či sú podporované všetky zadané atribúty, vzor aj akcie; vytvorí sa nové pravidlo, ktoré sa uloží do zoznamu pravidiel pre tabuľku `table_mark_id`; inicializuje sa tabuľka `table_mark_id` a nahrajú sa do nej všetky pravidlá vrátane nového pravidla; odomkne sa zámok. Ak počas uvedeného postupu nastane chyba, nové pravidlo ani tok nie sú vytvorené, odomkne sa zámok, a funkcia vráti `NULL`. Ak nastane chyba pri inicializácii tabuľky a pokuse vložiť všetky pravidlá, odstráni sa nové pravidlo a prebehne ešte jeden pokus o vloženie existujúcich pravidiel. Ak ani druhý pokus nie je úspešný, tabuľka ostane v nedefinovanom stave až do ďalšieho úspešného pokusu o vytvorenie alebo zrušenie pravidla.

Každý podporovaný tok musí mať nastavený atribút `ingress`. Žiadny iný atribút nie je podporovaný a nesmie byť nastavený. Podporované položky vo vzore a ich masky sú tie, ktoré sú uvedené v tabuľke 6.1. Okrem toho je ešte podporovaná meta položka `END`, ktorá značí koniec zoznamu položiek vo vzore. Prvou položkou vo vzore musí byť vždy `ETH`. Pre každú podporovanú položku sú definované podporované položky, ktoré za ňou môžu nasledovať, a masky. Pri kontrole vzoru je pri každej položke overované, či sa nasledujúca položka môže nachádzať za súčasnou položkou, a či zvolená maska je podporovaná. Opakovanie žiadnej položky nie je podporované. Ak prebehne kontrola položky úspešne, do štruktúry `rte_flow` sú uložené hodnoty a masky položky, a flag určujúci, že sa daná položka vo vzore nachádza. Keď všetky položky vzoru spĺňajú podmienky, vzor je podporovaný a do nového pravidla sa nastavia kľúče. Kľúče sú názvy políčok, nad ktorými sa vykonáva overovanie zhody, očakávané hodnoty a masky.

Po kontrole všetkých položiek vzoru nasleduje kontrola akcií. Podporované sú akcie `MARK`, `RSS`, `VOID` a `END`. Akcia `END` značí koniec zoznamu akcií. Akcia `VOID` je prázdna akcia, ktorá sa môže nachádzať na ľubovoľnom mieste v zozname a jednoducho sa vynecháva. V zozname akcií sa vždy musia nachádzať obe akcie `RSS` aj `MARK`. Pre akciu `RSS` je podporovaná len východzia konfigurácia, čo znamená, že položka `rss_conf` udávajúca konfiguráciu RSS musí byť nastavená na `NULL`. Prítomnosť oboch akcií `RSS`, `MARK` a číslo značky sa ukladá do štruktúry `rte_flow`. Ak prebehla kontrola akcií úspešne, do nového pravidla je pridaná akcia a parameter s číslom značky. Následne sa môže pristúpiť k vloženiu nového pravidla do tabuľky.

Rušenie pravidiel Pri rušení pravidla sú vykonané nasledujúce kroky: zamkne sa zámok v štruktúre `p4_device`; vymaže sa daný tok zo zoznamu tokov v súkromnej štruktúre zariadenia; inicializuje sa tabuľka `table_mark_id` a znovu sa do nej vložia všetky ostatné pravidlá; odomkne sa zámok. Ak nastala chyba pri inicializácii tabuľky alebo vkladaní pravidiel, tabuľka ostáva v nedefinovanom stave až do opätovného úspešného vytvorenia alebo zrušenia pravidla. Rušenie pravidiel je implementované funkciou `szedata2_flow_destroy()`.

Kapitola 7

Výsledky

S naimplementovaným systémom, ktorý je popísaný v kapitole 6, boli uskutočnené merania výkonnosti. Táto kapitola popisuje testovacie prostredie, spôsob jeho prípravy, testované prípady a konfigurácie, a na záver dosiahnuté výsledky a ich vyhodnotenie. Sledovaná bola rýchlosť prenosu paketov pri topológiách P2P a PVP s rôznymi konfiguráciami OVS pri využití offloadu pravidiel do karty, a bez využitia offloadu.

Cieľom je použitie vytvoreného systému na nízkoprofilových kartách COMBO-200G2QL. V čase písania tejto práce ešte nebol dokončený firmvér pre karty COMBO-200G2QL[1] obsahujúci aplikačné jadro, ktoré je možné generovať z popisu v jazyku P4. Preto boli merania vykonané so staršou kartou COMBO-100G2Q. Z dôvodu obmedzených dostupných hardvérových prostriedkov boli vykonané testy s vlastnou konfiguráciou, ktorá je popísaná v tejto kapitole, a nie s využitím frameworku VSPERF. Aby bolo možné naplno saturovať linku, bol použitý hardvérový generátor paketov Spirent. K dispozícii však neboli licencie pre využitie REST API na ovládanie Spirenta a automatizovaných testov priepustnosti, ktoré VSPERF používa. Taktiež karta COMBO-100G2Q sa z pohľadu softvéru v DPDK javí ako jeden port, ktorý musel byť pri konfiguráciách P2P aj PVP použitý ako vstupný aj výstupný port.

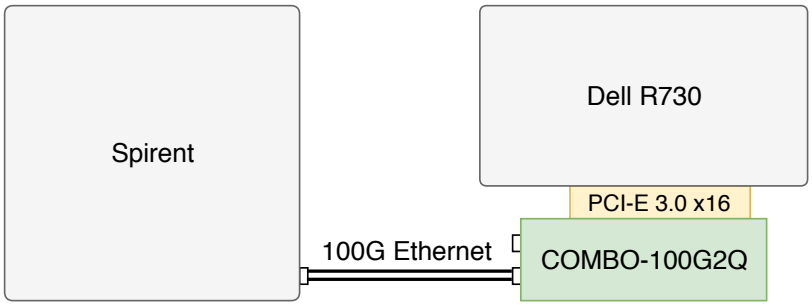
7.1 Testovacie prostredie

Testovanie prebiehalo na stroji Dell PowerEdge R730 s verziou BIOSu 1.0.4. Stroj obsahoval jeden osemjadrový procesor Intel® Xeon® E5-2630 v3 so základnou pracovnou frekvenciou 2,4 GHz. HyperThreading bol vypnutý. Osadených bolo osem pamäťových modulov DIMM DDR4 s registrom[2] s kapacitou 8 GiB a frekvenciou 2133 MHz. Použitá bola sieťová karta COMBO-100G2Q, pre ktorú boli prichystané firmvéry s 2, 4 a 8 DMA kanálmi v každom smere. Jeden fyzický port karty bol pripojený k hardvérovému generátoru paketov Spirent pomocou 100G optickej linky. Karta bola do stroja pripojená svojím bifurkovaným PCI-Express gen. 3 x16 slotom. Prepojenie testovacieho stroja a generátoru paketov znázorňuje schéma na obrázku 7.1.

Na stroji bol nainštalovaný operačný systém Scientific Linux verzia 7.4 (Nitrogen) s verziou jadra 3.10.0-514.26.2.el7.x86_64. Pre použitie karty COMBO-100G2Q boli ďalej potrebné moduly jadra `combo6core`, `combov3`, `szedata2`, `szedata2_cv3` súhrnne označované

[1] Celkový firmvér nie je obsahom tejto práce. Súčasťou práce je popis spracovania paketov v jazyku P4, z ktorého je generované aplikačné jadro pre spracovanie paketov.

[2] Angl. *registered memory*. Pamäťový modul obsahuje register medzi DRAM modulom a pamäťovým radičom.



Obr. 7.1: Schéma testovacieho zapojenia. Testovací stroj je prepojený jednou 100G linkou s generátorom paketov.

ako *combo ovládače*), knižnice `libcommlbr`, `libcombo`, `libfdt`, `libsze2`, `libp4dev` a balíček nástrojov `netcope-common`. K virtualizácii bol použitý emulátor QEMU a hypervízor KVM. Vo virtuálnom stroji bol nainštalovaný systém CentOS verzia 7.5 s jadrom 3.10.0-862.el7.x86_64. V tabuľke 7.1 je prehľad verzií použitého softvéru.

Tabuľka 7.1: Verzie softvéru na testovacom stroji.

Softvér	Verzia
balíček netcope-common	0.1.3
combo ovládače	0.9.11
libcommlbr	2.4.3
libcombo	1.8.7
libfdt	1.4.6
libsze2	1.1.7
libp4dev	commit 12c38da161ac
OVS	commit f82b3b6a2f4d + patche pre podporu rte_flow
DPDK	17.11 + úprava PMD szedata2 pre podporu rte_flow
QEMU	2.11

7.2 Príprava testovacieho prostredia

Pre účely testovania boli vytvorené inštalačné a konfiguračné skripty, ktoré automatizujú proces inštalácie potrebného softvéru a závislostí, a taktiež uľahčujú konfiguráciu základného nastavenia OVS a virtuálneho stroja, ktoré boli použité pri testovaní a meraní výkonnosti. Popis skriptov a ich použitia je v prílohe B. Pred testovaním bol skriptami na stroji nainštalovaný potrebný softvér:

```
./install_ovs.sh /home/local/xvidom00/sources
./install_qemu.sh /home/local/xvidom00/sources
```

Po naštartovaní systému boli na stroji pripravené obrovské stránky a naštartovaný démon `libvirtd` slúžiaci pre ovládanie virtuálneho stroja:

```
./post_boot_prepare.sh
```

Virtuálny stroj s názvom `pvp_vm` bol vytvorený a pripravený pre použitie skriptom:

```
./vm_create.sh /home/local/xvidom00/image pvp_vm
```

Skript vytvára obraz disku vo formáte `qcow2` pomocou nástroja `virt-builder`. Spúšťanie, zastavovanie, pripájanie sa k virtuálnemu stroju cez konzolu umožňuje nástroj `virsh`. Pomocou nástroja `virt-customize` sú do virtuálneho stroja prenášané súbory z fyzického stroja. Vo virtuálnom stroji bolo nainštalované DPDK aj s potrebnými závislosťami. Pre jednoduché automatické použitie pri meraniach bolo nastavené automatické prihlasovanie užívateľa `root` po naštartovaní virtuálneho stroja a spúšťanie skriptu štartujúceho DPDK aplikáciu `testpmd`. Inštaláciu a konfiguráciu vo virtuálnom stroji zabezpečil skript `vm_install_dpdk.sh`.

7.3 Merania výkonnosti

Výkonnosť akcelerovaného OVS bola odmeraná pre topológie P2P a PVP. Zhodnotenie výsledkov pre obe topológie nasleduje v podsekciách 7.3.1 a 7.3.2. Meraná bola rýchlosť prenosu paketov pri plnom zaťažení linky pre OVS s DPDK, kde bol pre fyzické rozhranie použitý DPDK ovládač szedata2 pre sieťovú kartu COMBO-100G2Q.

Dátový tok bol generovaný po dobu 10 s. Z hodnôt čítačov počtu vygenerovaných paketov (*tx_frames*), počtu prijatých paketov (*rx_frames*) na Spirentovi a známeho maximálneho počtu paketov za sekundu (*max_pps*) pre danú veľkosť ethernetového rámca na L2 vrstve (*packet_size*) bola podľa vzťahu 7.1 vypočítaná rýchlosť prenosu paketov (*packet_rate*) udávaná v počte paketov za sekundu (*p/s*):

$$packet\_rate = max\_pps * \frac{rx\_frames}{tx\_frames} [p/s] \tag{7.1}$$

Maximálny počet paketov za sekundu (*max_pps*) je vypočítaný podľa vzťahu 7.2, kde *L1_bitrate* je prenosová rýchlosť Ethernetu v bitoch za sekundu, *ifg* je medzera medzi rámcami na L1 vrstve (12 B) a *preamble* je preambula rámcu na L1 vrstve (8 B).

$$max\_pps = \frac{L1\_bitrate}{(packet\_size + ifg + preamble) * 8} [p/s] \tag{7.2}$$

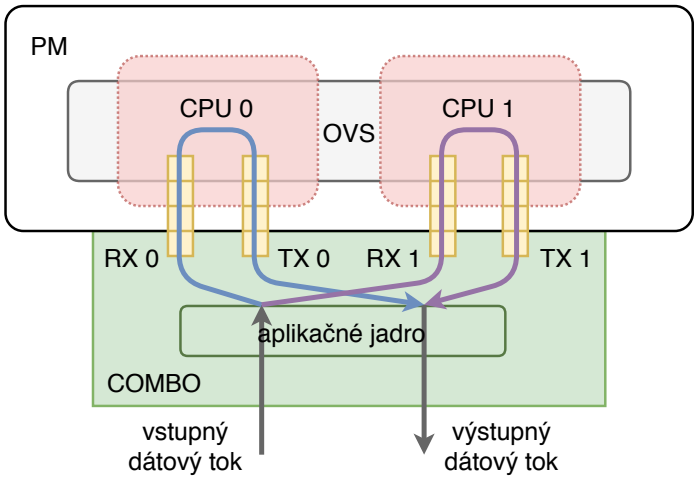
Potom napríklad pre najkratšie rámce (64 B na L2 vrstve) je maximálny počet paketov za sekundu na 100G Ethernete vypočítaný ako:

$$max\_pps = \frac{100 * 10^9}{(64 + 12 + 8) * 8} [p/s] = 148809523,8 [p/s]$$

Tabuľka 7.2: Súhrn permutácií vlastností meraných konfigurácií.

Topológia	Počet jadier CPU	Počet tokov	Offload do HW
P2P	1, 2, 4, 8	1, 30, 100	`true`, `false`
PVP	1, 2, 4	1, 30, 100	`true`, `false`

Merania boli vykonané pre rôzny počet použitých jadier CPU, počet tokov (*flows*) nastavených pre OVS, s aktivovaným a deaktivovaným offloadom do hardvéru. Pre každú konfiguráciu bola zmeraná rýchlosť prenosu paketov pre paketové dĺžky 64 B, 128 B, 256 B,



Obr. 7.2: Príklad rozdelenia obsluhy front v OVS na jadrá CPU pri topológii P2P. Fyzické jadro 0 obsluhuje jednu dvojicu RX a TX front. Fyzické jadro 1 obsluhuje druhú dvojicu.

512 B, 1024 B, 1280 B a 1518 B. Zhrnutie permutácií vlastností meraných konfigurácií je v tabuľke 7.2.

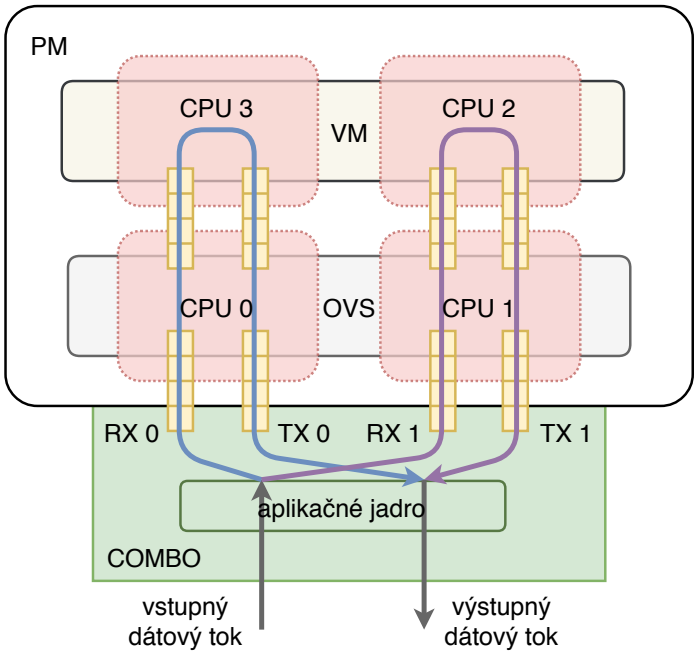
Počet jadier znamená, že bol použitý daný počet RX a TX front pre fyzické rozhranie, resp. aj pre virtuálne rozhranie pri topológii PVP. Na jednom jadre CPU bola obsluhovaná jedna RX a TX fronta fyzického, resp. aj virtuálneho rozhrania. Pri topológii PVP bolo virtuálnemu stroju pridelených toľko jadier fyzického CPU, koľko malo virtuálne rozhranie front. Vo virtuálnom stroji obsluhovalo každú dvojicu RX a TX front jedno jadro virtuálneho CPU, ktoré bolo pripnuté na pridelené jadro fyzického CPU. Príklad priechodu dát cez OVS a rozloženia na jadrá je znázornený na obrázkoch 7.2 (topológia P2P) a 7.3 (topológia PVP).

Počet tokov určuje počet rôznych OpenFlow tokov, ktorými bol OVS nakonfigurovaný. Pre počet tokov N bolo pri topológii P2P vytvorených N tokov s rôznou zdrojovou IP adresou, ktoré mali nastavený ako vstupný port fyzické rozhranie a ako akciu výstup na vstupné rozhranie. Napríklad pre fyzické rozhranie `dpdk0` bol tok so zdrojovou IP adresou 192.85.1.1 vytvorený nasledovne:

```
ovs-ofctl add-flow br0 \
    "in_port=dpdk0,dl_src=\"*\",dl_type=0x0800,nw_src=192.85.1.1,\
    action=output:in_port"
```

Pri topológii PVP bolo vytvorených N dvojíc tokov s rôznou zdrojovou IP adresou, kde jeden tok mal ako vstup nastavené fyzické rozhranie a ako akciu výstup na virtuálne rozhranie, a druhý tok mal ako vstup nastavené virtuálne rozhranie, a akcia bol výstup na fyzické rozhranie. Príklad vytvorenia dvojice tokov so zdrojovou IP adresou 192.85.1.1 pre fyzické rozhranie `dpdk0` a virtuálne rozhranie `dpdkv0`:

```
ovs-ofctl add-flow br0 \
    "in_port=dpdk0,dl_src=\"*\",dl_type=0x0800,nw_src=192.85.1.1,\
    action=output:dpdkv0"
ovs-ofctl add-flow br0 \
    "in_port=dpdkv0,dl_src=\"*\",dl_type=0x0800,nw_src=192.85.1.1,\
    action=output:dpdk0"
```



Obr. 7.3: Príklad rozdelenia obsluhy front v OVS na jadrá CPU pri topológii PVP. Fyzické jadro 0 obsluhuje jednu dvojicu RX a TX front fyzického rozhrania a jednu dvojicu front virtuálneho rozhrania (na strane hostiteľského systému). Fyzické jadro 1 obsluhuje ďalšie dvojice front fyzického a virtuálneho rozhrania. Jadrá 2 a 3 sú pridelené virtuálnemu stroju a na každom beží spracovanie jednej dvojice RX a TX front virtuálneho rozhrania na strane virtuálneho stroja.

Na Spirentovi bolo nastavené generovanie paketov s meniacou sa zdrojovou IP adresou v rozsahu 192.85.1.1 - 192.85.1.N. Firmvér rozdeľuje pakety do DMA front podľa hash funkcie vypočítanej zo všetkých podporovaných políčok hlavičiek protokolov. Keby sa v paketoch menila iba zdrojová IP adresa, pre nízke N mohlo vo firmvéri dochádzať k nerovnomernému rozdeleniu paketov do DMA front. Preto bola nastavená aj zmena zdrojovej MAC adresy, ktorá sa menila náhodne a tak bolo zabezpečené dostatočne rovnomerné rozloženie paketov do DMA front.

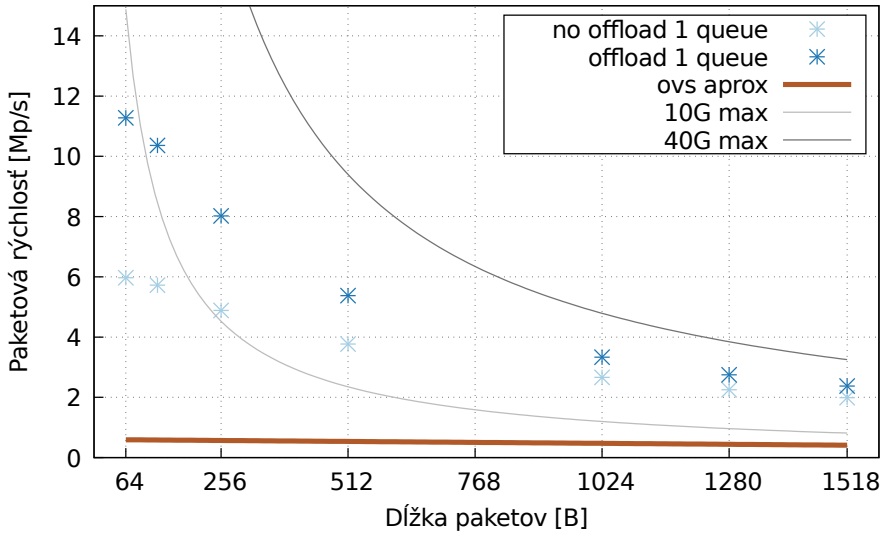
Rozloženie paketov do DMA front podľa výsledku hash funkcie nie je konfigurovateľné a za maximálnu hodnotu výsledku hash funkcie určujúceho výslednú DMA frontu je vždy považovaný celkový počet DMA front. Preto pri generovaní paketov uvedeným spôsobom s firmvérom obsahujúcim 8 DMA front a použití nižšieho počtu front by nebol celý dátový tok smerovaný do používaných front. Pri dlhších paketoch by to znamenalo, že by používané fronty dokázali spracovávať všetky prichádzajúce pakety aj pri plnom zaťažení linky, keďže do daných front by bola smerovaná iba časť dát, a nebolo by možné určiť limitnú hodnotu. Aby bolo možné odmerať maximálnu zvládnuteľnú rýchlosť, boli použité viaceré firmvéry s rôznym počtom DMA front. Pri meraniach s 1 a 2 frontami bol použitý firmvér, ktorý obsahoval 2 DMA fronty, pri meraniach so 4 frontami firmvér obsahujúci 4 DMA fronty, a pri meraniach s 8 frontami obsahoval firmvér 8 DMA front.

Príklad konfigurácie OVS je ukázaný v prílohe C. K spúšťaniu a konfigurácii OVS boli pri testoch použité skripty `p2p_setup_write.sh`, `p2p_setup_boot.sh`, `pvp_run_write.sh` a

`pvp_run_boot.sh`, ktoré sú popísané v prílohe B. Spustenie generovania paketov zo Spirenta bolo vykonané cez aplikáciu Spirent TestCenter Application.

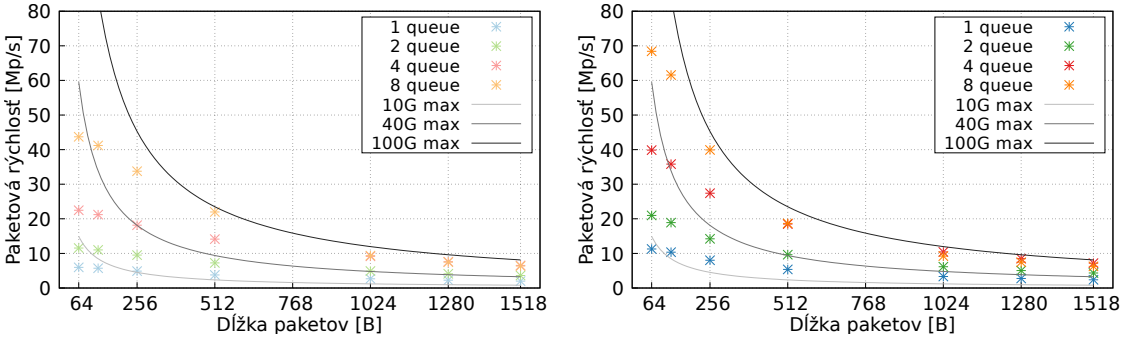
7.3.1 Výsledky pre topológiu P2P

Graf na obrázku 7.4 zobrazuje porovnanie rýchlosti prenosu paketov v topológii P2P na karte COMBO-100G2Q s využitím jedného jadra CPU pri použití jedného toku s aktivovaným offloadom a bez aktivovaného offloadu do hardvéru. V grafe je pre porovnanie znázornená aj priepustnosť klasického OVS nameraná v testoch od Intelu [17]. Bez offloadu do hardvéru bola pre najkratšie pakety dosiahnutá rýchlosť 5,97 Mp/s, čo je približne 5,3-krát viac ako s klasickým OVS. S offloadom do hardvéru to bolo 11,27 Mp/s, čo je asi 10,1-krát viac ako s klasickým OVS. S offloadom je dosahovaná o 88 % vyššia rýchlosť ako bez offloadu. S rastúcou veľkosťou paketov sa rozdiel vyrovnáva, čo je spôsobené tým, že v DPDK ovládači szedata2 sa pakety musia kopírovať z DMA buffrov do buffrov v DPDK. Pri dlhších paketoch tak pri rovnakom množstve kopírovaných dát je potrebné klasifikovať menej paketov a klasifikácia, ktorá je optimalizovaná offloadom do hardvéru tak predstavuje menšiu časť z celkového času potrebného na spracovanie paketu. Pre najdlhšie pakety je oproti klasickému OVS bez offloadu rýchlosť vyššia asi 2,4-krát vyššia. S offloadom je rýchlosť vyššia asi 2,9-krát oproti klasickému OVS a o 19 % vyššia ako bez offloadu.



Obr. 7.4: Topológia **P2P** - rýchlosť prenosu paketov cez OVS s kartou COMBO-100G2Q na **jednom jadre CPU**, **jeden tok**. Porovnanie so zapnutým offloadom do hardvéru (*offload 1 queue*) a bez offloadu (*no offload 1 queue*). Hnedá čiara (*ovs aprox*) zobrazuje pre porovnanie priepustnosť klasického OVS z testov realizovaných Intelom [17].

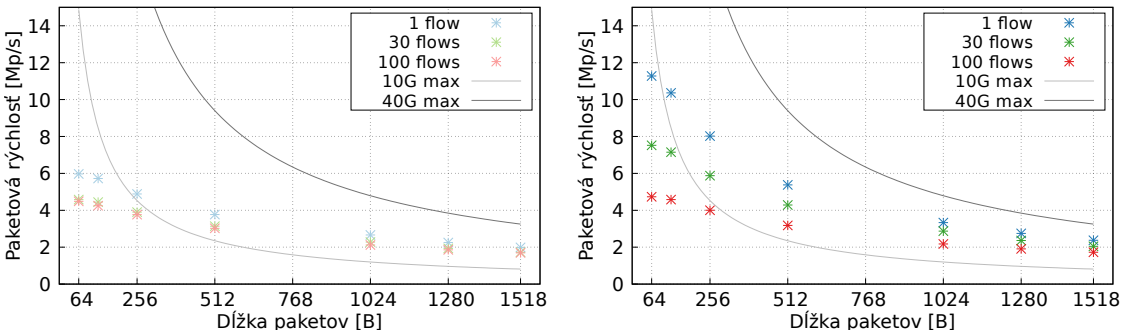
V grafoch na obrázku 7.5 je zobrazené ako sa zvyšuje rýchlosť prenosu paketov so zvyšujúcim sa počtom jadier CPU spracovávajúcich pakety pre jeden tok s offloadom aj bez offloadu. Pre najkratšie pakety je bez offloadu s 8 jadrami 7,3-krát vyššia rýchlosť ako s jedným jadrom. Pre najdlhšie pakety je to 3,2-krát viac. S offloadom je s 8 jadrami rýchlosť vyššia 6-krát oproti jednému jadru pre najkratšie pakety a 2,6-krát vyššia pre najdlhšie pakety. Pri použití 8 jadier CPU pri najkratších paketoch predstavuje dosiahnutá rýchlosť 45,9 % rýchlosti 100G linky, bez offloadu 29,3 %.



(a) **Bez offloadu** (b) **S offloadom do HW**

Obr. 7.5: Topológia P2P - porovnanie rýchlosti prenosu paketov cez OVS s kartou COMBO-100G2Q na **rôznom počte jadier CPU** (front), jeden tok.

Na obrázku 7.6 sú grafy zobrazujúce vplyv počtu tokov na rýchlosť prenosu paketov pri použití jedného jadra CPU s offloadom aj bez offloadu. S rastúcim počtom tokov sa zmenšuje výhoda použitia offloadu. Kým je v hardvéri voľné miesto pre nahratie pravidiel, je s offloadom dosahovaná vyššia rýchlosť ako bez offloadu. Testovací firmvér umožňoval nahrať 31 pravidiel. Pre vyšší počet tokov už je dosahovaná rovnaká rýchlosť s offloadom aj bez offloadu, čo je možné pozorovať na hodnotách pre 100 tokov. Pre kartu COMBO-100G2Q bolo problematické preložiť firmvér podporujúci vyšší počet pravidiel. V praktickom použití je však potrebné podporovať čo najväčší počet pravidiel, aby bolo možné maximálne využiť potenciál offloadu do hardvéru. V prípade karty COMBO-200G2QL je možné predpokladať, že počet podporovaných pravidiel bude väčší, nakoľko obsahuje väčší FPGA čip s lepšími parametrami.



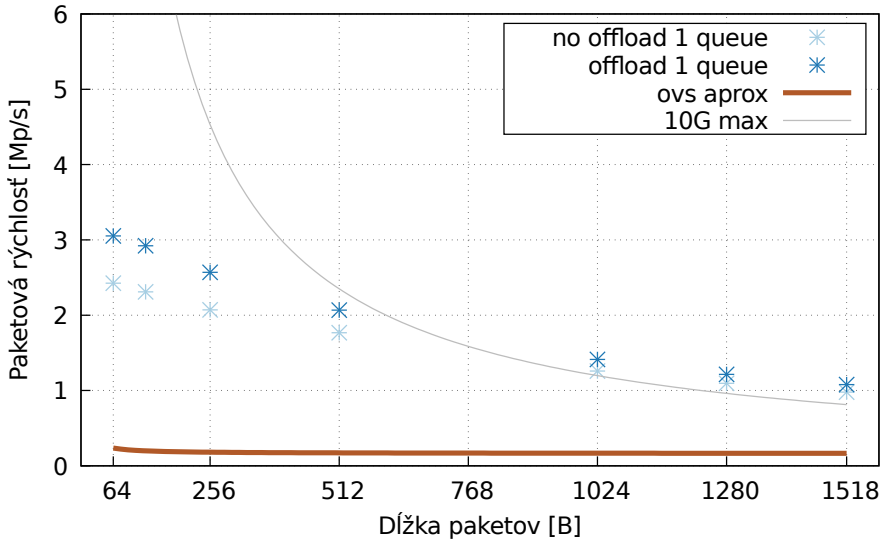
(a) **Bez offloadu** (b) **S offloadom do HW**

Obr. 7.6: Topológia P2P - porovnanie rýchlosti prenosu paketov cez OVS s kartou COMBO-100G2Q s **rôznym počtom tokov**, na jednom jadre CPU.

7.3.2 Výsledky pre topológiu PVP

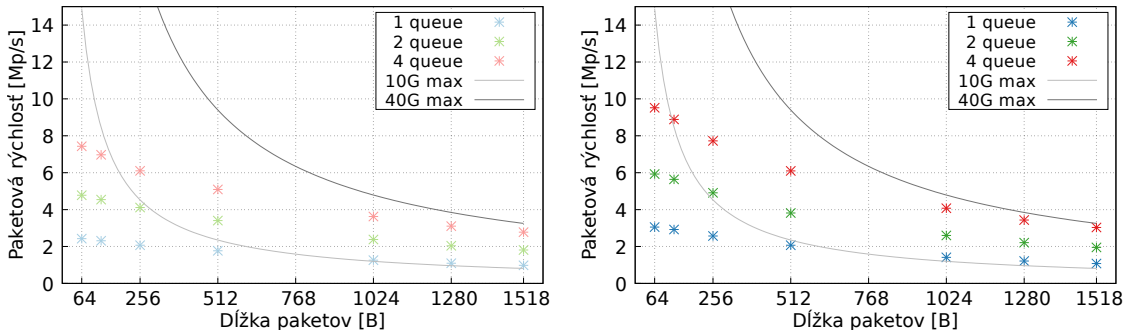
Na obrázku 7.8 je graf zobrazujúci porovnanie rýchlosti prenosu paketov v topológii PVP s kartou COMBO-100G2Q na jednom jadre CPU pri použití jedného toku s offloadom a bez offloadu do hardvéru. Pre porovnanie je v grafe zobrazená aj priepustnosť klasického OVS prevzatá z testov od Intelu [17]. Bez offloadu bola pre najkratšie pakety dosiahnutá rýchlosť

2,42 Mp/s, čo je asi 5,1-krát viac ako s klasickým OVS. S offloadom bola dosiahnutá rýchlosť 3,05 Mp/s, čo je približne 6,4-krát viac ako s klasickým OVS a o 26 % viac ako bez offloadu. V topológii PVP je vplyv offloadu výrazne nižší, pretože jadro obsluhujúce fronty sieťových rozhraní obsluhuje oproti topológii P2P navyše aj fronty virtuálneho rozhrania, a teda pri topológii PVP predstavuje offloadovaná klasifikácia z fyzického rozhrania menšiu časť celkového spracovania paketu ako pri topológii P2P. Pre najdlhšie pakety je s offloadom rýchlosť vyššia už len o 10 % ako bez offloadu. V porovnaní s topológiou P2P sa tiež prejavuje vplyv kopírovania paketov, ale pri topológii PVP sú pakety navyše kopírované aj pri prechode medzi hostiteľským a virtuálnym strojom.



Obr. 7.7

Obr. 7.8: Topológia **PVP** - rýchlosť prenosu paketov cez OVS s kartou COMBO-100G2Q na **jednom jadre CPU**, **jeden tok**. Porovnanie so zapnutým offloadom do hardvéru (*offload 1 queue*) a bez offloadu (*no offload 1 queue*). Hnedá čiara (*ovs aprox*) zobrazuje pre porovnanie priepustnosť klasického OVS z testov realizovaných Intelom [17].

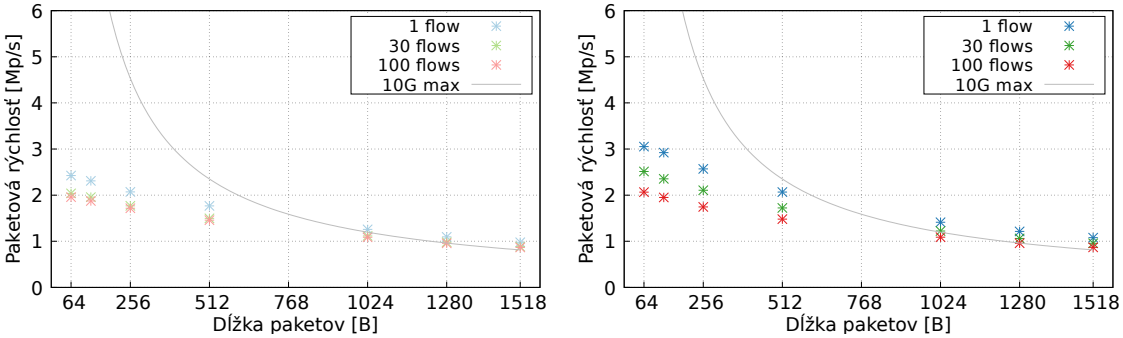


(a) **Bez offloadu** (b) **S offloadom do HW**

Obr. 7.9: Topológia PVP - porovnanie rýchlosti prenosu paketov cez OVS s kartou COMBO-100G2Q na **rôznom počte jadier CPU** (front), jeden tok.

Grafy na obrázku 7.9 zobrazujú ako sa mení celková rýchlosť prenosu paketov pri väčšom počte jadier CPU, ktoré spracúvajú pakety, pre jeden tok s offloadom aj bez offloadu. So 4 jadrami bez offloadu bola pre najkratšie pakety dosiahnutá 3-krát vyššia rýchlosť ako s jedným jadrom, pre najdlhšie pakety 2,8-krát viac ako s jedným jadrom. S offloadom bola pre najkratšie pakety dosiahnutá so 4 jadrami 3,1-krát vyššia rýchlosť ako s jedným jadrom. Pre najdlhšie pakety bola rýchlosť so 4 jadrami 2,8-krát vyššia ako s jedným jadrom. Prírastok rýchlosti pridaním jadier CPU je narozdiel od topológie P2P rovnaký s offloadom aj bez offloadu.

V grafoch na obrázku 7.10 je zobrazený vplyv počtu tokov na rýchlosť prenosu paketov pri použití jedného jadra CPU s offloadom a bez offloadu. Aj pre topológiu PVP platí to isté, čo je popísané pri topológii P2P. Keď počet tokov presiahne podporovaný počet pravidiel v hardvéri, stráca sa výhoda vyššej rýchlosti s použitím offloadu.



(a) **Bez offloadu** (b) **S offloadom do HW**

Obr. 7.10: Topológia PVP - porovnanie rýchlosti prenosu paketov cez OVS s kartou COMBO-100G2Q s **rôznym počtom tokov**, na jednom jadre CPU.

Kapitola 8

Záver

Cieľom práce bolo naštudovať princíp činnosti virtuálneho prepínača Open vSwitch (OVS), možnosti akceleračných kariet COMBO, a na základe analýzy súčasného spôsobu akcelerácie OVS pomocou frameworku DPDK navrhnúť a implementovať riešenie akcelerácie OVS s kartami COMBO. Výkonnosť implementovaného riešenia bola odmeraná a vyhodnotená pre rôzne konfigurácie OVS.

Práca začína popisom OVS a jeho štruktúry, kde sú popísané základné komponenty OVS a spôsob ich interakcie. Neskôr sú popísané akceleračné karty COMBO s dôrazom na nízkoprofilovú kartu COMBO-200G2QL. Predstavený je aj jazyk P4, ktorý umožňuje popisovať linku pre spracovanie paketov sieťovým zariadením. Pre karty COMBO je možné generovať firmvér z popisu v jazyku P4 s využitím prekladača z P4 do VHDL.

Ďalej sú popísané princípy frameworku DPDK a základné knižnice, ktoré sú použité v OVS využívajúcom DPDK. Dátová cesta OVS s DPDK spracúva kompletne všetky pakety v užívateľskom priestore narozdiel od štandardného OVS, v ktorom je dátová cesta implementovaná v module jadra. Dostupné sú viaceré správy z meraní výkonnosti klasického OVS a OVS s DPDK. Výsledky z týchto správ boli zhrnuté a popísané. Okrem akcelerácie OVS s použitím frameworku DPDK sa viacerí výrobcovia inteligentných sieťových kariet ako napríklad Cavium, Broadcom, Mellanox Technologies a Netronome snažia o prenesenie spracovania paketov do sieťových kariet. Tento prístup sa označuje ako hardvérová akcelerácia OVS. Popísané sú tri prístupy hardvérovej akcelerácie: beh OVS priamo v sieťovej karte, čiastočné a plné prenesenie spracovania tabuliek tokov z OVS do sieťovej karty (*offload*).

Akcelerácia OVS s kartami COMBO prebiehala v dvoch krokoch. Prvým krokom bolo použitie DPDK ovládača szedata2 pre karty COMBO a využitie akcelerovaného riešenia OVS s DPDK. V rámci práce bola do DPDK ovládača szedata2 pridaná podpora pre nízkoprofilovú kartu COMBO-200G2QL.

V druhom kroku bolo využité čiastočné prenesenie spracovania tabuliek tokov do karty COMBO. Od júna 2017 je súčasťou OVS aj rozhranie pre nahrávanie pravidiel z tabuliek tokov do sieťovej karty. Implementácia tohto rozhrania pre dátovú cestu cez DPDK v súčasnosti nie je súčasťou aktuálnej verzie OVS. Avšak spoločnosti Napatech a Mellanox Technologies implementovali sadu patchov pre OVS, ktoré využívajú DPDK rozhranie rte_flow k offloadu klasifikácie paketov do sieťovej karty. Pakety sú v sieťovej karte klasifikované a označené značkou, podľa ktorej sú potom v dátovej ceste s paketom vykonané príslušné akcie ako napríklad doručenie na cieľové rozhranie. Jedná sa o čiastočný offload do hardvéru. V rámci práce bol navrhnutý popis linky pre spracovanie paketov v jazyku P4, z ktorého môže byť pomocou prekladača z P4 do VHDL vygenerovaný firmvér pre karty

COMBO[1]. V čase písania práce ešte nebol hotový firmvér podporujúci aplikačné jadro generované z jazyku P4 pre nízkoprofilové karty COMBO-200G2QL, preto bola použitá karta COMBO-100G2Q. Do DPDK ovládača szedata2 bola naimplementovaná podpora rozhrania rte_flow, ktoré umožňuje nahrávať do karty klasifikačné pravidlá.

Na záver bola zmeraná a vyhodnotená výkonnosť OVS využívajúceho DPDK s kartou COMBO-100G2Q s klasifikáciou v karte aj bez klasifikácie v karte v dvoch topológiách zapojenia: fyzické rozhranie – fyzické rozhranie (P2P), fyzické rozhranie – virtuálny stroj – fyzické rozhranie (PVP). Meraná bola rýchlosť prenosu paketov pre rôzne dĺžky paketov. Pre každú topológiu boli vykonané merania vo viacerých konfiguráciách. S topológiou P2P bola pri použití jedného toku s najkratšími rámcami na jednom jadre CPU dosiahnutá rýchlosť prenosu paketov 5,97 Mp/s bez klasifikácie v karte (asi 5,3-krát viac ako klasické OVS) a 11,27 Mp/s s klasifikáciou v karte (asi 10,1-krát viac ako klasické OVS). S klasifikáciou v karte bola dosiahnutá o 88 % vyššia rýchlosť ako bez klasifikácie v karte. S topológiou PVP pri rovnakej konfigurácii ako je uvedené pre topológiu P2P bola dosiahnutá rýchlosť 2,42 Mp/s bez klasifikácie v karte (asi 5,1-krát viac ako klasické OVS) a 3,05 Mp/s s klasifikáciou v karte (asi 6,4-krát viac ako klasické OVS). S klasifikáciou v karte bola rýchlosť vyššia o 26 % ako bez klasifikácie. S väčšou dĺžkou paketov sa rozdiely oproti klasickému OVS aj rozdiely medzi variantami s klasifikáciou a bez klasifikácie v karte zmenšujú pre obe topológie P2P, aj PVP.

Možností pokračovania práce je niekoľko. V práci bol použitý DPDK ovládač pre DMA prenosy cez rozhranie SZE2. Pre karty COMBO však vzniká aj nový typ DMA prenosov založený na princípe deskriptorov, ktorý je prirodzený pre spôsob prenosu paketov v DPDK. Použitie DMA prenosov na princípe deskriptorov nevyžaduje v DPDK ovládači kopírovanie obsahu paketov tak, ako je to u ovládača szedata2, vďaka čomu je možné dosahovať vyššiu priepustnosť s nižším počtom jadier CPU. Avšak maximálna priepustnosť je nižšia ako s rozhraním SZE2. V práci bola porovnaná výkonnosť deskriptorových DMA prenosov s prenosmi cez rozhranie SZE2. Firmvér pre deskriptorové prenosy v súčasnosti nie je stabilný a je vo vývoji. Pri nasadení OVS je však dôležitým kritériom znižovanie záťaže CPU, preto pri akcelerácii OVS môže byť v budúcnosti vhodné použiť firmvér s deskriptorovými DMA prenosmi.

Implementované riešenie bolo v práci testované na karte COMBO-100G2Q, pre ktorú je možné podporovať iba nízky počet klasifikačných pravidiel, čo je pre reálne použitie nedostatočné. Po dokončení príslušného firmvéru pre nízkoprofilovú kartu COMBO-200G2QL sa počíta s použitím vytvoreného riešenia aj pre túto kartu. Karta COMBO-200G2QL obsahuje väčší FPGA čip, preto sa dá predpokladať, že počet podporovaných klasifikačných pravidiel bude väčší. Ďalšou možnosťou je optimalizácia firmvérových komponent tak, aby bol počet podporovaných pravidiel čo najväčší.

V komunite OVS a DPDK prebieha vývoj knižníc pridávajúcich aj podporu plného offloadu do hardvéru. Riešenie akcelerácie OVS implementované v tejto práci využíva čiastočný offload do hardvéru. V budúcnosti bude možné doplniť aj podporu plného offloadu.

[1]Dielom tejto práce je len P4 program popisujúci linku pre spracovanie paketov, nie kompletný firmvér.

Literatúra

[1] Benáček, P.; Puš, V.; Kubátová, H.: P4-to-VHDL. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, IEEE, 2016, ISBN 978-1-5090-2356-1, s. 148–155, doi:10.1109/FCCM.2016.46.
URL http://ieeexplore.ieee.org/document/7544769/

[2] Bodireddy, B.; Fischetti, A.: OVS-DPDK Datapath Classifier. 2016, [cit. 2018-05-18].
URL
https://software.intel.com/en-us/articles/ovs-dpdk-datapath-classifier

[3] Bodireddy, B.; Fischetti, A.: OvS-DPDK Datapath Classifier – Part 2. 2016, [cit. 2018-05-18].
URL https:
//software.intel.com/en-us/articles/ovs-dpdk-datapath-classifier-part-2

[4] Bradner, S.: Benchmarking Terminology for Network Interconnection Devices. RFC 1242 (Informational), Júl 1991, updated by RFC 6201.
URL http://www.ietf.org/rfc/rfc1242.txt

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

[6] Broadcom: *PS225*. [cit. 2018-05-18].
URL https:
//www.broadcom.com/products/ethernet-connectivity/network-adapters/ps225

[7] Cavium: *LiquidIO OVS Software Architecture*. 2017, [cit. 2018-01-10].
URL http://www.cavium.com/Documents/WhitePapers/Adapters/
WP_LiquidIO_OVS_Software_Architecture.pdf

[8] CESNET: *Cards*. [cit. 2017-12-31].
URL https://www.liberouter.org/technologies/cards/

[9] Chandran, S.: Enabling hardware acceleration in OVS-DPDK using DPDK Framework. 2017, [cit. 2018-01-12].
URL https://www.slideshare.net/LF_OpenvSwitch/lfovs17enabling-hardware-acceleration-in-ovsdpdk-using-dpdk-framework

[10] Chiosi, M.; Clarke, D.; Willis, P.; aj.: *Network Functions Virtualisation – Introductory White Paper*. ETSI, 2012, [cit. 2017-12-30].
URL https://portal.etsi.org/nfv/nfv_white_paper.pdf

[11] Cisco: *Cisco Nexus 1000V Series Virtual Switch Data Sheet*. 2016, [cit. 2017-12-30].
URL https://www.cisco.com/c/en/us/products/collateral/switches/nexus-1000v-switch-vmware-vsphere/data_sheet_c78-492971.html

[12] Cisco: *Features and Scalability Information for Cisco Nexus 1000V*. 2016, [cit. 2017-12-30].
URL https://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus1000/nexus1000v_feature_matrix/Cisco_Nexus_1000V_Features_Scalability_Info.html

[13] Emmerich, P.; Raumer, D.; Gallenmüller, S.; aj.: Throughput and Latency of Virtual Switching with Open vSwitch. *Journal of Network and Systems Management*: s. –, ISSN 1064-7570, doi:10.1007/s10922-017-9417-0.
URL http://link.springer.com/10.1007/s10922-017-9417-0

[14] Emmerich, P.; Raumer, D.; Wohlfart, F.; aj.: Performance characteristics of virtual switching. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, IEEE, 2014, ISBN 978-1-4799-2730-2, s. 120–125, doi:10.1109/CloudNet.2014.6968979.
URL http://ieeexplore.ieee.org/document/6968979/

[15] Han, B.; Gopalakrishnan, V.; Ji, L.; aj.: Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, ročník 53, č. 2, 2015: s. 90–97, ISSN 0163-6804, doi:10.1109/MCOM.2015.7045396.
URL http://ieeexplore.ieee.org/document/7045396/

[16] Horman, S.: OvS Hardware Offload with TC Flower. 2017, [cit. 2018-01-12].
URL https://www.slideshare.net/LF_OpenvSwitch/lfovs17ovs-hardware-offload-with-tc-flower

[17] Intel: *Intel® Open Network Platform Release 2.1 Performance Test Report*. 2016, [cit. 2018-05-12].
URL https://download.01.org/packet-processing/ONPS2.1/Intel_ONP_Release_2.1_Performance_Test_Report_Rev1.0.pdf

[18] Kutch, P.; Johnson, B.: *SR-IOV for NFV Solutions*. Intel, 2017, [cit. 2018-05-18].
URL https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/sr-iov-nfv-tech-brief.pdf

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

[20] Mellanox Technologies: *Mellanox ASAP^2: Accelerated Switching and Packet Processing*. 2017, [cit. 2018-01-10].
URL http://www.mellanox.com/related-docs/products/SB_asap2.pdf

[21] Netcope Technologies: *Netcope FPGA Boards*. [cit. 2017-12-31].
URL https://www.netcope.com/en/products/fpga-boards

[22] Netronome Systems: *Agilio® OVS Software Architecture*. 2016, [cit. 2018-01-12].
URL https://www.netronome.com/media/redactor_files/WP_Agilio_SW.pdf

[23] Netronome Systems: *OVS Test Plan*. 2016, [cit. 2018-01-12].
URL https://www.netronome.com/media/redactor_files/TP_OVS_Test_Plan.pdf

[24] Netronome Systems: *Open vSwitch Offload and Acceleration with Agilio® CX SmartNICs*. 2017, [cit. 2018-01-12].
URL https://www.netronome.com/media/documents/WP_OVS_Benchmarking.pdf

[25] Netronome Systems: *Programming Netronome Agilio® SmartNICs*. 2017, [cit. 2018-01-12].
URL
https://www.netronome.com/media/documents/WP_NFP_Programming_Model.pdf

[26] O'Mahonny, B.: The Open vSwitch Exact-Match Cache. 2017, [cit. 2018-05-18].
URL https:
//software.intel.com/en-us/articles/the-open-vswitch-exact-match-cache

[27] Open Networking Foundation: *Software-Defined Networking: The New Norm for Networks*. 2012, [cit. 2017-12-30].
URL https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[28] OPNFV: *CHARACTERIZE VSWITCH PERFORMANCE FOR TELCO NFV USE CASES LEVEL TEST REPORT*. 2016, [cit. 2017-12-30].
URL
https://artifacts.opnfv.org/vswitchperf/brahmaputra/report/report.pdf

[29] OPNFV: *VSperf Home: Test Methodology*. 2016, [cit. 2017-12-30].
URL https://wiki.opnfv.org/display/vsperf/Test+Methodology

[30] The P4 Language Consortium: *The P4 Language Specification*. 2017, [cit. 2018-05-15].
URL https://p4.org/p4-spec/p4-14/v1.0.4/tex/p4.pdf

[31] The P4 Language Consortium: *The $P4_{16}$ Language Specification*. 2017, [cit. 2018-05-15].
URL https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.pdf

[32] Paolino, M.; Fanguede, J.; Nikolaev, N.; aj.: Turning an open source project into a carrier grade vswitch for NFV. In *2016 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC)*, IEEE, 2016, ISBN 978-1-5090-1245-9, s. 22–27, doi:10.1109/ICNIDC.2016.7974529.
URL http://ieeexplore.ieee.org/document/7974529/

[33] Pfaff, B.; Davie, B.: The Open vSwitch Database Management Protocol. RFC 7047, December 2013, doi:10.17487/RFC7047.
URL https://rfc-editor.org/rfc/rfc7047.txt

[34] Pfaff, B.; Pettit, J.; Koponen, T.; aj.: The Design and Implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA: USENIX Association, 2015, ISBN 978-1-931971-218, s. 117–130.
URL https:
//www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[35] Rosenblum, M.; Waldspurger, C.: I/O Virtualization. *Queue*, ročník 9, č. 11, 2011-11-01: s. 30–, ISSN 15427730, doi:10.1145/2063166.2071256.
URL http://dl.acm.org/citation.cfm?doid=2063166.2071256

[36] Sans, F.; Gamess, E.: Analytical Performance Evaluation of Different Switch Solutions. *Journal of Computer Networks and Communications*, ročník 2013, 2013: s. 1–11, ISSN 2090-7141, doi:10.1155/2013/953797.
URL http://www.hindawi.com/journals/jcnc/2013/953797/

[37] Srinivasan, V.; Suri, S.; Varghese, G.: Packet Classification Using Tuple Space Search. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '99, New York, NY, USA: ACM, 1999, ISBN 1-58113-135-6, s. 135–146, doi:10.1145/316188.316216.
URL http://doi.acm.org/10.1145/316188.316216

[38] Tahhan, M.; O'Mahony, B.; Morton, A.: Benchmarking Virtual Switches in the Open Platform for NFV (OPNFV). RFC 8204, RFC Editor, September 2017.

[39] Vido, M.: *DPDK nad síťovými kartami COMBO [online]*. Bakalářská práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2016 [cit. 2018-01-15].
URL http://www.fit.vutbr.cz/study/DP/BP.php?id=18312

[40] VMware: *Overview of vNetwork Distributed Switch concepts*. 2015, [cit. 2017-12-30].
URL https://kb.vmware.com/s/article/1010555

[41] Wang, C.; Spatscheck, O.; Gopalakrishnan, V.; aj.: Toward High-Performance and Scalable Network Functions Virtualization. *IEEE Internet Computing*, ročník 20, č. 6, 2016: s. 10–20, ISSN 1089-7801, doi:10.1109/MIC.2016.111.
URL http://ieeexplore.ieee.org/document/7781548/

[42] Data Plane Development Kit. [cit. 2018-01-08].
URL http://www.dpdk.org

[43] Design Decisions In Open vSwitch. [cit. 2018-05-18].
URL http://docs.openvswitch.org/en/latest/topics/design/

[44] DPDK Network Interface Controller Drivers Release 17.11.0. 2017, [cit. 2018-01-08].
URL http://fast.dpdk.org/doc/pdf-guides-17.11/nics-17.11.pdf

[45] Generic flow API (rte_flow). [cit. 2018-05-15].
URL http://dpdk.org/doc/guides/prog_guide/rte_flow.html

[46] DPDK Programmer's Guide Release 17.11.0. 2017, [cit. 2018-01-08].
URL http://fast.dpdk.org/doc/pdf-guides-17.11/prog_guide-17.11.pdf

[47] DPDK Testpmd Application User Guide Release 17.11.0. 2017, [cit. 2018-01-08].
URL http://fast.dpdk.org/doc/pdf-guides-17.11/testpmd_app_ug-17.11.pdf

[48] Open vSwitch. [cit. 2017-12-31].
URL http://openvswitch.org/

[49] Open vSwitch database server manual. [cit. 2018-01-04].
URL http://openvswitch.org/support/dist-docs/ovsdb-server.1.txt

[50] P4. [cit. 2018-01-15].
URL https://p4.org/

[51] Porting Open vSwitch to New Software or Hardware. [cit. 2018-01-04].
URL http://docs.openvswitch.org/en/latest/topics/porting/

[52] Reference Guide. [cit. 2018-01-02].
URL http://docs.openvswitch.org/en/latest/ref/

[53] Why Open vSwitch? [cit. 2018-01-02].
URL http://docs.openvswitch.org/en/latest/intro/why-ovs/

Prílohy

Zoznam príloh

Príloha A

Obsah CD

Priložené CD obsahuje nasledujúce adresáre a súbory:

- `dp-xvidom00.pdf`

  Písomná správa vo formáte PDF.

- `README.txt`

  Súbor s popisom obsahu CD.

- `dpdk-patches/`

  V adresári sa nachádza patch obsahujúci všetky zmeny v ovládači szedata2 a v ostatných súboroch stromu zdrojových kódov DPDK verzie 17.11, ktoré sú potrebné pre správny beh ovládača szedata2 s rozšíreniami vykonanými v tejto práci. Tento patch využíva inštalačný skript v adresári `scripts/`.

- `ovs-patches/`

  V adresári sa nachádzajú patche pridávajúce do OVS podporu offloadu s využitím DPDK API `rte_flow`. Patche nie sú dielom autora tejto práce.

- `scripts/`

  Adresár obsahuje pomocné inštalačné a konfiguračné skripty použité pri testovaní OVS. Popis skriptov je v prílohe B.

- `src/dpdk/drivers/net/szedata2/`

  Adresár obsahujúci zdrojové súbory DPDK ovládača *szedata2*.

- `src/p4_firmware/ovs_offload/`

  Zdrojové kódy v jazyku P4 popisujúce aplikačné jadro firmvéru podporujúceho offload pravidiel z OVS do hardvéru.

- `text/`

  Zdrojové texty a obrázky písomnej správy pre LaTeX.

- `vm/`

  Adresár obsahuje konfiguračné XML súbory pre spustenie virtuálneho stroja použitého pri testovaní.

Príloha B

Inštalačné a konfiguračné skripty

V tejto prílohe sú popísané inštalačné a konfiguračné skripty, ktoré sa nachádzajú v adresári `scripts/` na priloženom CD.

`./install_ovs.sh <source_dir>`

Nainštaluje vývojové balíčky s nástrojmi, ovládačmi a knižnicami pre kartu COMBO. Stiahne repozitár obsahujúci zdrojové kódy knižnice `libp4dev` potrebnej na ovládanie aplikačného jadra firmvéru generovaného z P4. Stiahne repozitáre `dpdk` a `ovs`, a nainštaluje OVS s podporou DPDK zo zdrojových kódov. Na repozitáre `dpdk` a `ovs` aplikuje potrebné patche. Sťahované repozitáre sú uložené do adresára `<source_dir>`.

`./install_qemu.sh <source_dir>`

Nainštaluje balíčky s nástrojmi použitými pri práci s virtuálnym strojom a knižnicu `libvirt`. Stiahne repozitár `qemu` a nainštaluje QEMU zo zdrojových kódov. Sťahovaný repozitár je uložený do adresára `<source_dir>`.

`./load_firmware.sh [<firmware_path>]`

Nabootuje firmvér do karty COMBO. Ak je zadaný parameter, tak aj zapíše firmvér s názvom `<firmware_path>` do flash pamäte karty. Ak parameter nie je zadaný, tak sa iba nabootuje firmvér, ktorý bol naposledy zapísaný do flash pamäte karty.

`./post_boot_prepare.sh`

Pripraví obrovské stránky (*huge pages*), ktoré sú potrebné pre použitie DPDK a naštartuje démona `libvirtd`.

`./vm_create.sh <vm_dir> <vm_name>`

Kompletne pripraví nový virtuálny stroj pre testovanie PVP konfigurácie. Stroj dostane meno `<vm_name>` a obraz disku vo formáte `qcow2` je vytvorený v adresári `<vm_dir>`. K vytvoreniu stroja je použitý nástroj `virt-builder`. Pomocou nástroja `virt-customize` je do stroja prenesený skript `vm_install_dpdk.sh` pre inštaláciu DPDK. Potom je stroj spustený pomocou nástroja `virsh` a na stroji je nainštalované DPDK vrátane potrebných závislostí. V adresári `<vm_dir>` sú potom uložené aj konfiguračné XML súbory slúžiace pre spustenie stroja nástrojom `virsh`.

`./vm_install_dpdk.sh`

Vo virtuálnom stroji nainštaluje potrebné závislosti a DPDK. Nastaví automatické prihlásenie užívateľa `root` po štarte systému a spustenie skriptu `vm_run_testpmd.sh`, ktorý vo virtuálnom stroji spúšťa DPDK aplikáciu `testpmd`.

`./vm_run_testpmd_[1|2|4]q.sh`

Vo virtuálnom stroji spúšťa DPDK aplikáciu po štarte virtuálneho stroja s počtom front 1, 2 alebo 4 (podľa názvu skriptu). Pred spustením `testpmd` zabezpečuje pripravenie obrovských stránok vo virtuálnom stroji a pripojenie virtuálneho rozhrania na modul jadra `igb_uio`. Všetky varianty skriptu sú do virtuálneho stroja nakopírované pod rovnakým názvom `/root/vm_run_testpmd.sh`.

`./p2p_setup_base.sh <queue_count> <flow_count> <hw_offload> <emc>`

Konfiguruje topológiu P2P.

Nakonfiguruje OVS, fyzické DPDK rozhranie, spustí OVS a nastaví pravidlá pre toky. Parameter `<hw_offload>` nastavuje hodnotu konfiguračnej voľby OVS `hw-offload`, ktorej hodnota `false` znamená, že sa nebude používať offload pravidiel do hardvéru, a hodnota `true` znamená, že sa bude používať offload. Parameter `<emc>` nastavuje hodnotu konfiguračnej voľby OVS `emc-insert-inv-prob`, ktorá určuje ako často sa nový tok bude ukladať do Exact Match Cache (EMC) pamäte tokov. Hodnota N znamená, že jeden z N tokov sa uloží do EMC. Parameter `<queue_count>` nastavuje počet RX a TX front pre nakonfigurovanie DPDK rozhrania. Parameter `<flow_count>` určuje, koľko rôznych tokov (*flow*) bude vytvorených pre vstup fyzického rozhrania. V skripte je nastavená premenná `PCIDEV` podľa umiestnenia karty COMBO na zbernici PCI-Express. V prípade testovania na inom stroji je potrebné mať nastavenú správne túto premennú.

`./p2p_setup_boot.sh <queue_count> <flow_count> <hw_offload> <emc>`

Konfiguruje topológiu P2P.

Nabootuje firmvér a predá svoje parametre skriptu `p2p_setup_base.sh`

`./p2p_setup_write.sh <firmware> <queue_count> <flow_count> \`
`<hw_offload> <emc>`

Konfiguruje topológiu P2P.

Zapíše firmvér `<firmware>` do karty a nabootuje ho. Potom predá ostatné svoje parametre skriptu `p2p_setup_base.sh`

`./pvp_setup_base.sh <queue_count> <flow_count> <hw_offload> <emc>`

Konfiguruje topológiu PVP.

Konfiguruje OVS a fyzické rozhranie rovnako ako skript `p2p_setup_base.sh`, ale okrem toho nakonfiguruje aj virtuálne rozhranie. Potom spustí OVS a nastaví pravidlá pre toky. Význam parametrov je rovnaký ako u skriptu `p2p_setup_base.sh`. Rovnako tak treba mať správne nastavenú premennú `PCIDEV`.

`./pvp_setup_boot.sh <queue_count> <flow_count> <hw_offload> <emc>`

Konfiguruje topológiu PVP.

Nabootuje firmvér a predá svoje parametre skriptu `pvp_setup_base.sh`

`./pvp_setup_write.sh <firmware> <queue_count> <flow_count> \ <hw_offload> <emc>`

Konfiguruje topológiu PVP.

Zapíše firmvér `<firmware>` do karty a nabootuje ho. Potom predá ostatné svoje parametre skriptu `pvp_setup_base.sh`.

`./pvp_start_vm.sh <vm_dir> <vm_name> <queue_count>`

Spustí virtuálny stroj `<vm_name>`, ktorého obraz sa nachádza v adresári `<vm_dir>` a nastaví počet front virtuálneho rozhrania v danom stroji na `<queue_count>`. Skript používa konfiguračné XML súbory z adresáru `vm/` k spusteniu stroja. Do stroja je pred spustením pomocou nástroja `virt-customize` nahraný skript pre spúšťanie `testpmd`.

`./pvp_run_boot.sh <vm_dir> <vm_name> <queue_count> <flow_count> \ <hw_offload> <emc>`

Spája konfiguráciu OVS pre topológiu PVP a spustenie virtuálneho stroja. Príslušné parametre predáva skriptom `pvp_setup_boot.sh` a `pvp_start_vm.sh`.

`./pvp_run_write.sh <firmware> <vm_dir> <vm_name> <queue_count> \ <flow_count> <hw_offload> <emc>`

Spája konfiguráciu OVS pre topológiu PVP a spustenie virtuálneho stroja. Príslušné parametre predáva skriptom `pvp_setup_write.sh` a `pvp_start_vm.sh`.

Príloha C

Konfigurácia a spustenie OVS

V tejto prílohe sú príklady konfiguračných nastavení OVS a spúšťania, ktoré boli využité v skriptoch pri testovaní.

Konfigurácia a spustenie OVS:

```
# Spustenie ovsdb-server
/usr/local/share/openvswitch/scripts/ovs-ctl --no-ovs-vswitchd start
# Nastavenie pouzitia datovej cesty v uzivatelskom priestore cez DPDK
ovs-vsctl --no-wait set Open_vSwitch . \
    other_config:dpdk-init=true
# Spustenie ovs-vswitchd
/usr/local/share/openvswitch/scripts/ovs-ctl --no-ovsdb-server start
# Konfiguracia OVS
# Prepoklada sa ze su nastavene premenne:
# PMD_CPU_MASK ... obsahuje masku jadier CPU pouzitych pre PMD vlakna
# HW_OFFLOAD ... true aktivuje, false deaktivuje offload tokov do HW
# EMC_INV_PROB ... hodnota znamena, ze 1 z EMC_INV_PROB tokov bude
# ulozeny do EMC
# Priklad nastavenia premennych:
# PMD_CPU_MASK=0xff # jadra 0-7
# HW_OFFLOAD=true # pouzije sa offload
# EMC_INV_PROB=0 # do EMC sa neukladaju ziadne toky
ovs-vsctl --no-wait set Open_vSwitch . \
    other_config:dpdk-socket-mem=1024
ovs-vsctl --no-wait set Open_vSwitch . \
    other_config:pmd-cpu-mask=${PMD_CPU_MASK}
ovs-vsctl --no-wait set Open_vSwitch . \
    other_config:hw-offload=${HW_OFFLOAD}
ovs-vsctl --no-wait set Open_vSwitch . \
    other_config:emc-insert-inv-prob=${EMC_INV_PROB}
# Vytvorenie noveho bridgeu podporujuceho datovu cestu s DPDK
ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev
```

Konfigurácia fyzického DPDK rozhrania `dpdk0` s 1 RX a TX frontou pre kartu umiestnenú na PCI lokalite 0000:03:00.0 a priviazaním PMD vlákna 0 na jadro 1:

```
ovs-vsctl add-port br0 dpdk0 -- set Interface \
    dpdk0 type=dpdk options:dpdk-devargs="0000:03:00.0-port0"
ovs-vsctl --no-wait set Interface dpdk0 options:n_rxq=1
ovs-vsctl --no-wait set Interface dpdk0 options:n_txq=1
ovs-vsctl --no-wait set Interface dpdk0 other_config:pmd-rxq-affinity="0:1"
```

Konfigurácia virtuálneho DPDK rozhrania `dpdkv0` typu `dpdkvhostuser` s 1 RX a TX frontou a priviazaním PMD vlákna 0 na jadro 1:

```
ovs-vsctl add-port br0 dpdkv0 -- \
    set Interface dpdkv0 type=dpdkvhostuser
ovs-vsctl --no-wait set Interface dpdkv0 options:n_rxq=1
ovs-vsctl --no-wait set Interface dpdkv0 options:n_txq=1
ovs-vsctl --no-wait set Interface dpdkv0 other_config:pmd-rxq-affinity="0:1"
```

Reštart užívateľského démona `ovs-vswitchd`:

```
/usr/local/share/openvswitch/scripts/ovs-ctl --no-ovsdb-server restart
```

Zastavenie užívateľského démona `ovs-vswitchd`:

```
/usr/local/share/openvswitch/scripts/ovs-ctl --no-ovsdb-server stop
```

Vytvorenie N tokov pre topológiu P2P:

```
# Zmazanie existujucich tokov
ovs-ofctl del-flows br0
# Vytvorenie N novych tokov
for i in $(seq 1 $N); do
    ovs-ofctl add-flow br0 \
        "in_port=dpdk0,dl_src=\"*\",dl_type=0x0800,nw_src=192.85.1.$i,\
        action=output:in_port"
done
```

Vytvorenie N tokov pre topológiu PVP:

```
# Zmazanie existujucich tokov
ovs-ofctl del-flows br0
# Vytvorenie N novych tokov
for i in $(seq 1 $N); do
    ovs-ofctl add-flow br0 \
        "in_port=dpdk0,dl_src=\"*\",dl_type=0x0800,nw_src=192.85.1.$i,\
        action=output:dpdkv0"
    ovs-ofctl add-flow br0 \
        "in_port=dpdkv0,dl_src=\"*\",dl_type=0x0800,nw_src=192.85.1.$i,\
        action=output:dpdk0"
done
```