VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

TESTOVÁNÍ OPEN VSWITCH A DPDK

TESTING OPEN VSWITCH AND DPDK

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. OTTO ŠABART**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RUDOLF ČEJKA**
SUPERVISOR

BRNO 2017

Zadání diplomové práce/20053/2016/xsabar00

Vysoké učení technické v Brně - Fakulta informačních technologií

Centrum výpočetní techniky Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Šabart Otto, Bc.**
Obor: Počítačové sítě a komunikace
Téma: **Testování Open vSwitch a DPDK**
Testing Open vSwitch and DPDK
Kategorie: Počítačové sítě

Pokyny:

1. Prostudujte možnosti softwarového přepínače Open vSwitch.
2. Prostudujte možnosti akcelerace softwarového přepínače Open vSwitch pomocí technologie Data Plane Development Kit (DPDK).
3. Změřte výkonnost různých konfigurací Open vSwitch bez DPDK i s DPDK na použitém hardwaru.
4. Po dohodě s vedoucím připravte a integrujte testovací prostředí pro automatizované periodické testování výkonnosti Open vSwitch s DPDK.
5. Zhodnoťte dosažené výsledky a stanovte optimální režim Open vSwitch s DPDK pro různé aplikace.

Literatura:

- *HERBERT, Thomas. SDN, Openflow, and Open vSwitch Pocket Primer. Mercury Learning & Information, 2014. ISBN 9781937585457*
- *Dokumentace k Open vSwitch na http://openvswitch.github.io/support*
- *Dokumentace k DPDK na http://dpdk.org/doc*

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2 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: **Čejka Rudolf, Ing.**, CVT FIT VUT
Konzultant: Okuliar Adam, Ing., FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Práce popisuje virtuální přepínač Open vSwitch a jeho architekturu. Zabývá se jeho akcelerací - především pomocí knihovny Data Plane Development Kit (DPDK). Popisuje architekturu této knihovny, rozebírá její jednotlivé funkční celky a popisuje možnosti její konfigurace. Další část práce popisuje metodologii zvolenou pro testování výkonu virtuálních přepínačů. Tato metodologie byla následně využita pro návrh a implementaci prostředí pro plně automatizované testování výkonu přepínače Open vSwitch s DPDK s využitím automatizačních systémů Koji, Jenkins, Beaker a VSperf. Zároveň byly implementovány nástroje pro automatické porovnávání získaných výsledků. Celé vytvořené prostředí bylo následně použito pro změření výkonu několika základních konfigurací přepínače Open vSwitch, a to jak s využitím knihovny DPDK, tak i bez ní. Provedená měření jsou v práci zhodnocena a diskutována. Závěr práce se zabývá velkým množstvím rozšíření a vylepšení implementovaných testů.

Abstract

The project is about the virtual switch called Open vSwitch and its architecture. It deals with an acceleration of the switch mainly by using Data Plane Development Kit (DPDK). Furthermore, it describes the architecture of the DPDK kit and analyses the individual functional units. Furthermore, it describes the architecture of the DPDK kit, analyses the individual functional units and describes the possibilities of its configuration. Another part of the project describes the methodology chosen for a performance testing of virtual switches. Subsequently, this methodology was used to make a design and environment implementation for fully automatic Open vSwitch s DPDK performance testing with the use of automatic systems such as Koji, Jenkins, Beaker a VSperf. Simultaneously, the tools for automatic comparison of produced results were implemented. The created environment was then used for the performance measurement of several basic Open vSwitch configurations with and without the use of DPDK. The implemented measurements are discussed and evaluated in the project. The final project's stage provides a great amount of the enlargement and improvement of the implemented tests.

Klíčová slova

DPDK, Open vSwitch, virtuální přepínač, automatizace testování výkonu, orchestrace, zpracování výkonnostních výsledků

Keywords

DPDK, Open vSwitch, virtual switch, performance test automation, orchestration, performance results processing

Citace

ŠABART, Otto. *Testování Open vSwitch a DPDK*. Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Čejka Rudolf.

Testování Open vSwitch a DPDK

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Rudolfa Čejky. Další informace mi poskytl Ing. Adam Okuliar ze strany firmy Red Hat. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Otto Šabart
24. května 2017

Poděkování

Tímto bych rád poděkoval Ing. Adamovi Okuliarovi z firmy Red Hat a Ing. Rudolfovi Čejkovi za odbornou pomoc, ochotu a čas, který mi při vedení mé práce poskytli.

Obsah

Kapitola 1

Úvod

S nárůstem výkonu hardware a neustále rostoucími nároky na bezpečnost, rychlost a škálovatelnost systémů se stále častěji objevuje termín *virtualizace*. Virtualizace umožňuje přistupovat ke zdrojům v počítači jiným způsobem, než jakým fyzicky existují.

V poslední době je již virtualizace natolik rozšířená, že se stává standardem, bez kterého by mnoho institucí a firem ani nemohlo existovat. Virtualizace totiž firmám poskytuje velkou flexibilitu a především jim šetří nemalé peníze. Virtualizovat lze v dnešní době na mnoha různých úrovních. Lze virtualizovat datová úložiště, hardware nebo síť. Virtualizace hardware, kdy je vytvořen jeden nebo více virtuálních serverů (neboli hostů) chovajících se jako reálné fyzické servery s operačním systémem, je jedním z nejčastějších případů virtualizace [7][6].

Už zde začínají být patrné výhody virtualizovaného prostředí. Jeden výkonný fyzický server s nainstalovaným *hypervizorem*[1] může nahradit až několik dalších fyzických systémů tím, že jsou tyto systémy přesunuty do virtualizovaného prostředí. Díky tomu je výkon fyzického stroje využíván naplno. V případě, že bychom tyto systémy nevirtualizovali, museli bychom spravovat několik fyzických strojů. Za tyto všechny stroje bychom museli platit za jejich uložení do datového centra (tzv. server housing). Zároveň by ani nemusel být jejich potenciál plně využíván [22].

Kromě ušetření nákladů za server housing s sebou virtualizace přináší mnoho dalších výhod. Umožňuje šetřit náklady spojené s chlazením (více fyzických serverů potřebuje více chladit), nabízí rychlý *server provisioning*[2], zlepšení dostupnosti, zvýšení bezpečnosti nebo izolaci aplikací [19].

S virtuálními systémy přicházejí k řešení i nové problémy. Na virtuálních hostech chtějí mít zákazníci spuštěné služby stejně, jako kdyby byly spuštěné přímo na fyzickém serveru. Jedním z nejdůležitějších požadavků je správně nakonfigurovaná síť.

Díky virtualizaci se objevují pojmy jako snímky (z angl. snapshots) nebo živá migrace (z angl. live migration), u které je možné přesunout aktuální obraz virtuálního hosta z jednoho hypervizoru na druhý (například z důvodu poruchy zařízení, na kterém hypervizor běží, nebo údržby), a to nejen v rámci jednoho datového centra, ale i do datového centra umístěného na úplně jiném místě. Síťová lokalita virtuálního zařízení se tak může dynamicky měnit. Zároveň je zde ale požadavek, aby přesunutý virtuální host měl k dispozici

[1] Hypervizor - Software zajišťující běh virtualizovaných hostů, který řídí jejich přístup k fyzickému hardware počítače. Zároveň od sebe virtuální hosty odděluje. V některých částech práce je pod slovem hypervizor myšlen nejen SW zajišťující virtualizaci, ale i fyzický hardware, na kterém tento software běží.

[2] Server provisioning - https://en.wikipedia.org/wiki/Provisioning#Server_provisioning

správně nakonfigurované jak přepínaní, tak i směrování. Tedy, aby jeho připojení do sítě bylo na druhém hypervizoru funkční stejně dobře, jako tomu bylo na hypervizoru prvním.

1.1 Virtualizace síťových zařízení

Ve většině případů chceme mít virtuální hosty jak síťově propojené mezi sebou, tak i připojené do vnější sítě. Z tohoto důvodu bylo nutné vyvinout mechanismy pro jejich propojení. Byly vyvinuty tzv. *virtuální přepínače*, které jsou schopny přepínat rámce mezi virtuálními hosty jak v rámci jednoho hypervizoru, tak i mezi virtuálními a fyzickými zařízeními. Virtuálními přepínači se bude tato práce zabývat především. Práce se zabývá měřením jejich výkonu, akcelerací a automatizací jejich testování.

Virtualizace jde ale mnohem dále. Do virtualizovaných prostředí se nepřesouvají pouze přepínače, ale i další síťové prvky jako směrovače, firewally, vyvažovače zátěže (z angl. load balancers), systémy IDS[3], generátory provozu a další. Tato virtuální síťová zařízení představují tzv. *virtuální síťové funkce* (z angl. Network Function Virtualization - NFV). Virtuální síťové funkce mohou běžet na standardních serverech. Jeden server může obsluhovat i více různých síťových funkcí. Není potřeba mít pro každou síťovou funkci speciální hardwarové zařízení [39].

Aby virtuální funkce mohly prakticky nahradit specializovaná hardwarová zařízení, musí mít dostatečný výkon. Ačkoliv se vývojáři operačních systémů snaží, výkon v určitých případech užití dostatečný není. Tato práce vysvětluje proč tomu tak je a ukazuje řešení v podobě obcházení síťové vrstvy OS.

Pojem virtuálních síťových funkcí velmi úzce souvisí se softwarově definovanými sítěmi.

1.2 Softwarově definované sítě

V poslední době se stále více dostávají do popředí tzv. softwarově definované sítě (z angl. Software-defined networking - SDN). Tyto sítě mají administrátorům umožnit flexibilnější konfiguraci různých síťových topologií, především v datových centrech.

V klasickém síťovém zařízení je datová (z angl. data plane) i řídicí vrstva (z angl. control plane) součástí jednoho a toho samého zařízení. V softwarově definovaných sítích jsou tyto vrstvy oddělené. Datová vrstva provádějící přeposílání provozu běží přímo v hardware. Pravidla pro přeposílání ale získává z vrstvy řídicí. Řídicí vrstva má za úkol rozhodovat, jak bude s určitým síťovým tokem naloženo. Může řešit přeposílání rámců, směrování, definovat pravidla pro filtrování provozu apod.

Toto rozdělení dává administrátorům velkou flexibilitu v možnosti nastavení chování celé sítě. SDN jim totiž umožňuje jednotlivé síťové prvky konfigurovat z jednoho centrálního místa (tzv. SDN řadiče) [16].

[3] IDS - Intrusion Detection System - systém pro odhalení průniku.

Kapitola 2

Síťové přepínače

Síťový přepínač je v terminologii počítačových sítí aktivní prvek pracující především na druhé síťové vrstvě modelu OSI[1] (občas může částečně zasahovat i do vrstev vyšších), propojující dohromady další zařízení v rámci sítě. Síťové přepínače můžeme rozdělit do dvou kategorií dle úrovně fungování - na fyzické přepínače (bare-metal) a na virtuální (fungující v software).

2.1 Fyzické přepínače

Pro propojení fyzických zařízení (serverů, routerů, jiných přepínačů apod.) slouží fyzické přepínače. Ještě do nedávna fyzické přepínače představovaly jedinou možnost propojení fyzických i virtuálních zařízení na linkové vrstvě [28]. V dnešní době stále zastávají funkci propojení fyzických zařízení, vznikla ale potřeba propojit zařízení virtuální. Zde nastupují právě přepínače virtuální.

Nespornou výhodou fyzických přepínačů je rychlost. Té dosahují implementací přepínání přímo do hardwarových ASIC čipů. Nevýhodou tohoto přístupu je cena vývoje takového zařízení, která se projevuje i na ceně hotového produktu. Další zásadní nevýhodou je fakt, že především díky ASIC čipům jsou fyzické přepínače jednoúčelová zařízení. Pokud je potřeba implementovat novou funkcionalitu, je ve většině případů nutné obstarat celé nové zařízení.

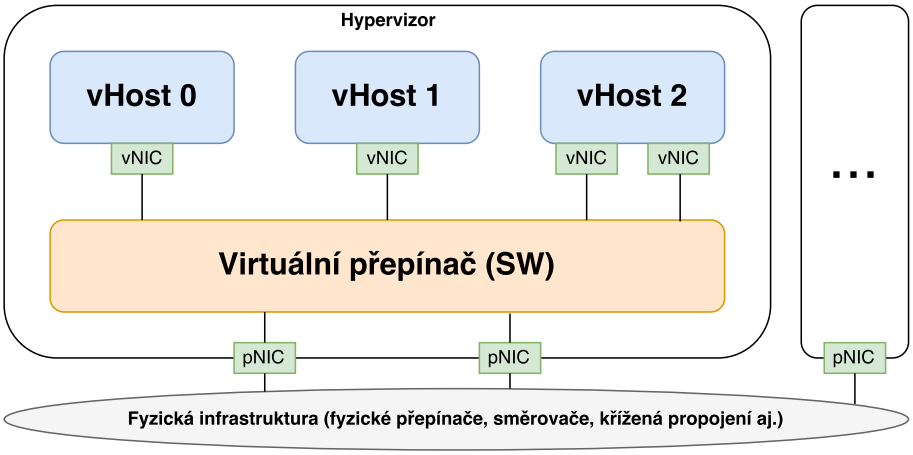
2.2 Virtuální přepínače

Jeden z prvních přístupů propojení virtuálních hostů s okolním světem zajišťovaly fyzické přepínače. Využívalo se zde víceportových síťových karet, kde každý z portů byl přiřazen jednomu nebo více virtuálním hostům. Hosti pak k těmto portům přistupovali přímo pomocí technologie zvané *PCI device passthrough*. Tato technologie virtuálním hostům umožňuje přistupovat k zařízením připojeným k PCI sběrnici *přímo*, bez jakéhokoliv zásahu hostujícího systému (hypervizoru). Přepínání zajišťoval externí fyzický přepínač, což nemuselo být vždy výhodné. Síťové rámce odeslané z hostů totiž musely pokaždé opustit hypervizor, i když hosti komunikovali v rámci stejné sítě. V dnešní době se od tohoto přístupu upouští a vznikají daleko univerzálnější řešení. Jako alternativa vznikají *virtuální přepínače*.

Virtuální přepínač je programová vrstva, která se nachází na úrovni hostujícího systému (hypervizoru) zajišťujícího běh virtuálních hostů [28].

[1] Open Systems Interconnection model

Na obrázku 2.1 lze vidět typické použití virtuálního přepínače v rámci hypervizoru. Virtuální hosti (vHost) jsou k virtuálnímu přepínači připojeny pomocí jejich logických (virtuálních) sítových karet (vNIC). Samotný virtuální přepínač pak s vnějším světem komunikuje pomocí fyzických sítových karet (pNIC). Tyto karty (přesněji jejich rozhraní) jsou do přepínače připojeny také [10].



Obrázek 2.1: Typická aplikace virtuálního přepínače.

Úkolem virtuálního přepínače je zajistit L2 konektivitu jak mezi různými virtuálními hosty v rámci virtuální infrastruktury (přepínání mezi vNIC-vNIC), tak zajistit i konektivitu mezi virtuální a fyzickou infrastrukturou (přepínání vNIC-pNIC), například za účelem směrování [10].

2.2.1 Problémy spojené s virtuálními přepínači

I když se na první pohled může zdát, že virtuální přepínače s sebou přinášejí samé výhody, objevují se i nové problémy. Virtuální přepínače na hypervizorech s sebou přinášejí další vrstvu, kterou správci sítí musí udržovat. Objevují se i výkonnostní problémy. S přibývajícím počtem virtuálních hostů neroste pouze zátěž na hypervizor ale roste i zátěž na virtuální přepínač. Vznikají tak další technologie, které mají přispět k jednodušší správě, větší bezpečnosti a lepší škálovatelnosti.

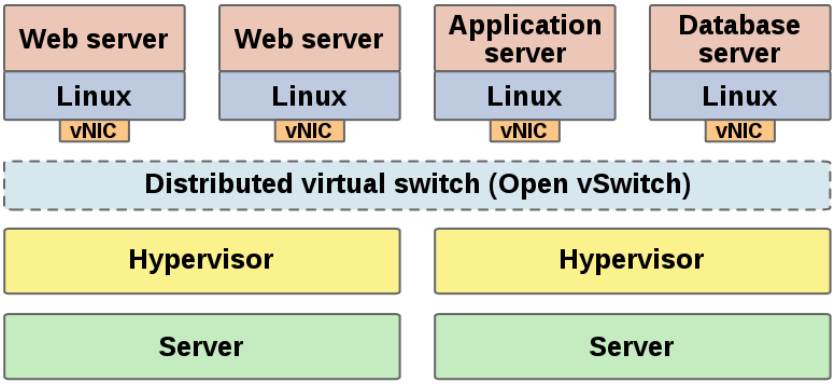
Distribuované přepínače

Jednou z technologií sloužící k jednodušší (centralizované) správě je koncept tzv. *distribuovaných přepínačů*, neboli DVS[2]. Jak lze vidět na obrázku architektury 2.2, distribuovaný přepínač se posouvá ještě o úroveň dále. Datová a řídicí vrstva přepínače jsou zde *odděleny*. To umožňuje centrálně spravovat jednotlivé datové vrstvy různých virtuálních přepínačů (i v rámci různých hypervizorů) centrálně z jednoho externího řadiče (např. pomocí protokolu OpenFlow[3]). Tento externí řadič tak představuje řídicí vrstvu přepínače.

[2]DVS - Distributed Virtual Switching

[3]OpenFlow - Komunikační protokol umožňující vzdálený přístup k datové vrstvě přepínače nebo směrovače.

Dokonce se nemusíme omezovat pouze na ovládání datových vrstev virtuálních přepínačů. I mnoho fyzických přepínačů dnes umožňuje vzdálenou správu datové vrstvy pomocí protokolu OpenFlow. Dostáváme se tak k SDN (viz 1.2).



Obrázek 2.2: Znázornění architektury distribuovaného přepínače.

2.2.2 Linux-Bridge

Operační systém GNU/Linux umožňuje vytvořit speciální rozhraní, které může být použito pro vytvoření virtuálního přepínače. V terminologii operačního systému GNU/Linux se toto rozhraní nazývá *bridge*. Linuxový bridge je nekomplexní virtuální přepínač operující pouze na vrstvě L2.

Pro konfiguraci linuxového bridge jsou zapotřebí nástroje *bridge-utils*[4], běžící v uživatelském prostoru. Kromě bridge-utils lze také použít nástroje z projektu *iproute2*.

Pro základní propojení několika virtuálních hostů mezi sebou v rámci jednoho hypervizoru, popř. s okolním světem, je linuxový bridge plně dostačující řešení. Pro pokročilejší použití už bohužel dostačující není. Řešení zmiňovaná dále jsou flexibilnější a mají o mnoho více funkcí [10].

2.2.3 Cisco Nexus 1000V

Jednou ze společností, která přišla s vlastní proprietární implementací virtuálního přepínače, je společnost Cisco. Virtuální přepínač se jmenuje *Cisco Nexus 1000V* a jedná se o komerční produkt [4]. Hlavní nevýhodou je jeho vysoká cena, která může dosahovat až k částkám okolo 21000$ [5].

2.2.4 Open vSwitch (OvS)

Mnoho společností si vytvořilo vlastní implementace virtuálních přepínačů pro své proprietární produkty. Open vSwitch (zkráceně OvS), je ale příklad otevřeného virtuálního přepínače. Jeho první verze byla vydána již v roce 2009. Od té doby je projekt aktivně dále udržován a rozšiřován o nové funkce. Kód je licencován pod Apache License, Verze 2[6]. Celý

[4] Nástroje *bridge-utils* lze najít v repozitáři na adrese https://git.kernel.org/cgit/linux/kernel/git/shemminger/bridge-utils.git

[5] https://buildprice.cisco.com/N1000V/buyn1k

[6] https://www.apache.org/licenses/LICENSE-2.0

je naprogramován v jazyce C, přičemž je zde důraz na kód nezávislý na platformě. Může tak být jednoduše přenesen na jiná prostředí. Oficiálně je podporovaný na systémech GNU-/Linux, FreeBSD, NetBSD, Windows a vznikají verze také pro hypervizory VMware ESXi a Microsoft Hyper-V [24].

Cílem vývojářů Open vSwitch je implementovat takový přepínač, který by byl kvalitou hodný nasazení do produkčních prostředí. Zároveň se vývojáři snaží vytvořit přepínač, který bude možné spravovat pomocí standardizovaných rozhraní programově [34], především v SDN sítích. To se vývojářům daří, protože dnes je jedním z nejpoužívanějších virtuálních přepínačů. Lze ho používat na většině linuxových systémů díky přímé podpoře v linuxovém jádře. Spolupracuje s většinou hypervizorových i kontejnerových systémů jakými jsou Xen, KVM nebo Docker. Má také výbornou podporu ve virtualizačním API *libvirt*[7] [10][24]. Dále je nasazován ve velkých cloudových systémech jakými jsou OpenStack[8] nebo OpenNebula[9] [28].

Mezi aktuálně podporované technologie patří služby zajišťující bezpečnost, monitorování, QoS (Quality of Service) nebo automatizovanou správu [34]. Viz například:

- virtuální LAN (VLAN) - 802.1Q
- bonding s nebo bez LACP (Link Aggregation Control Protocol)
- NetFlow, sFlow(R), IPFIX
- QoS (Quality of Service), policing
- IPsec, GENEVE, GRE, GRE over IPsec, VxLAN, STT a LISP tunely
- OpenFlow 1.0
- SNMP

Open vSwitch mimo jiné podporuje i protokol GRE[10]. Díky němu lze například propojit několik různých sítí napříč několika různými datovými centry [34].

Přepínač dále obsahuje podporu pro ACL[11], pomocí kterých umožňuje filtrovat provoz na vrstvách L3 a L4.

Už zde je vidět, že tento přepínač nelze porovnávat s Linux-Bridge. Open vSwitch na rozdíl od Linux-Bridge dokáže částečně operovat i na vrstvách L3 a L4 a už v základu obsahuje podporu pro protokoly, které u Linux-bridge musí být řešeny dodatečnými nástroji [25].

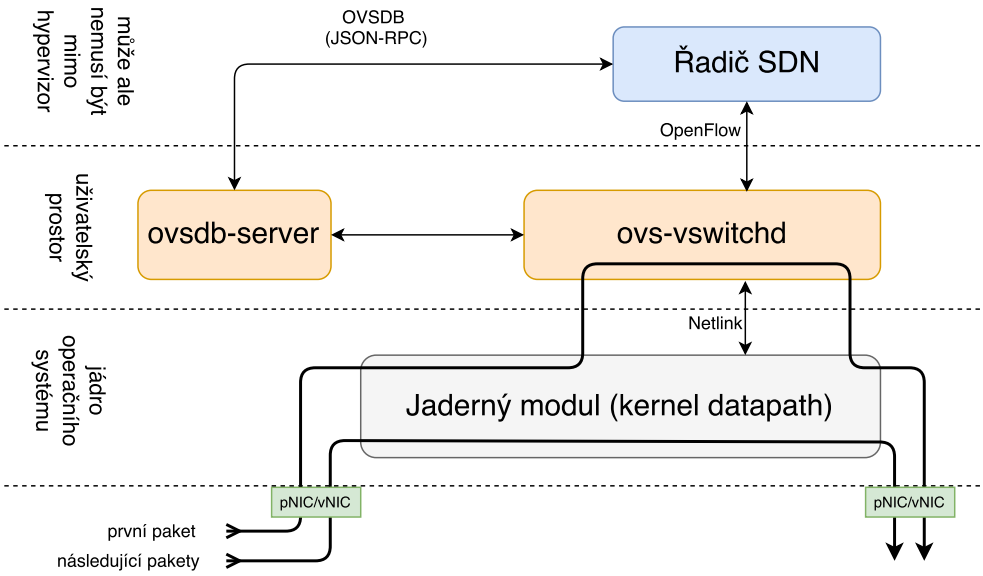
Jeho největší výhodou je ale jeho otevřený kód a multiplatformnost. V porovnání s uzavřenými virtuálními přepínači, které dokáží pracovat vždy pouze v jednom prostředí, je prostředí běhu Open vSwitch vybíráno samotným uživatelem. Ten si může sám vybrat operační systém i hypervizor, na kterém bude přepínač provozovat. Díky tomu je Open vSwitch velice modulární, přenositelný a jednoduše rozšiřitelný.

[7] libvirt - https://libvirt.org
[8] OpenStack - https://www.openstack.org
[9] OpenNebula - https://opennebula.org
[10] GRE - Generic Routing Encapsulation - protokol určený k zapouzdření paketů jednoho protokolu do druhého. Využívá se především k tzv. tunelování - u VPN (Virtual private network), přenosu IPv6 v síti IPv4 apod.
[11] ACL (Access Control Lists) - seznamy pravidel umožňující filtrování provozu.

Architektura virtuálního přepínače

Přepínač lze používat ve dvou různých módech. Prvním z nich je klasický *learning mód*, ve kterém se Open vSwitch chová jako standardní přepínač. To znamená, že přepínání rámců se provádí na základě naučených položek v tabulce CAM[12]. Hlavním účelem vzniku Open vSwitch je ale fungování v módu druhém. V tomto módu lze chování přepínače kompletně programovat pomocí pravidel protokolu OpenFlow.

Celý systém přepínače se skládá ze dvou hlavních komponent. První komponentou je démon běžící v uživatelském prostoru zvaný `ovs-vswitchd`. Druhou částí je jaderný modul představující datovou vrstvu přepínače. Obrázek 2.3 znázorňuje, jak spolu tyto dvě komponenty spolupracují při zpracování příchozích paketů.



Obrázek 2.3: Základní architektura přepínače Open vSwitch.

Modul datové vrstvy přepínače, který se nachází přímo v linuxovém jádře (od verze 3.10), přijme paket z virtuálního (vNIC) nebo fyzického (pNIC) rozhraní karty. V případě, že už datová vrstva byla pomocí `ovs-vswitchd` nakonfigurována pro rozpoznání daného typu paketu, zachová se jednoduše podle nakonfigurovaných instrukcí. Těmto instrukcím se také říká *akce*. Akce mohou být různého typu. Základní akcí je přeposlání paketu na jiné rozhraní, popř. tunel. Mezi další akce pak patří modifikace paketů, vzorkování paketů nebo jejich zahazování.

V druhém případě, kdy datová vrstva pro daný typ paketu akci nenalezne, předá paket `ovs-vswitchd` do uživatelského prostoru. Ten musí rozhodnout, jak s paketem naloží. Pokud se paket rozhodne zahodit, nemusí dělat nic. Pouze paket odstraní ze své paměti. Pokud se s paketem rozhodne vykonat nějakou z dříve zmíněných akcí, předá ho zpět datové vrstvě i s příslušnou informací o akci. Většinou `ovs-vswitchd` instruuje datovou vrstvu ještě k tomu, aby si akci uložila do své interní cache a automaticky ji vykonávala pro podobné pakety (bez potřeby `ovs-vswitchd`) [24].

[12]Content-addressable memory

Jak lze vidět na obrázku 2.3, pravidla pro rozhodování získává `ovs-vswitchd` z SDN řadiče. Rozhodovací pravidla jsou součástí tzv. OpenFlow tabulek, které jsou z tohoto řadiče získávány pomocí protokolu OpenFlow.

Dalšími důležitými částmi Open vSwitch jsou [35]:

- `ovsdb-server` – Program poskytující JSON-RPC[13] rozhraní pro přístup k jedné nebo více Open vSwitch databázím. Prostřednictvím tohoto démona získává `ovs-vswitchd` konfiguraci z databáze.
- `ovsdb-tool` – Nástroj pro správu databázových souborů. Tento nástroj neslouží ke správě databáze jako takové, slouží pouze k manipulaci s databázovými soubory. K přímé manipulaci s běžící databází slouží `ovsdb-client`.
- `ovsdb-client` – Slouží pro manipulaci s běžící databází. Při každém dotazu se připojuje k běžícímu databázovému serveru (`ovsdb-server`).
- `ovs-dpctl` – Nástroj pro vytváření, modifikaci nebo odstranění datových cest (datapaths) v jaderném modulu.
- `ovs-vsctl` – Umožňuje získávání a aktualizaci konfigurace `ovs-vswitchd`.
- `ovs-appctl` – Program sloužící ke konfiguraci chování běžících démonů, například konfiguraci logování. Každý z démonů přijímá stejnou obecnou množinu příkazů. Kromě této množiny příkazů pak každý ještě může přijímat příkazy speciální.
- `ovs-ofctl` – nástroj pro monitorování a administraci OpenFlow přepínačů a řadičů. Umožňuje zobrazit aktuální stav OpenFlow přepínače včetně aktuální konfigurace nebo položek v OpenFlow tabulce. Umožňuje práci s jakýmkoliv OpenFlow přepínačem, ne pouze s Open vSwitch.
- `ovs-pki` – Tato utilita slouží pro správu PKI[14] (pro OpenFlow).
- `ovs-testcontroller` – Jednoduchá implementace OpenFlow řadiče spravujícího libovolný počet přepínačů přes protokol OpenFlow. Tento řadič slouží především pro testovací účely a neměl by se používat v produkčním prostředí.

Přepínač Open vSwitch dokáže pracovat kompletně v *uživatelském prostoru* - pro svou funkčnost nepotřebuje zavedený jaderný modul implementující datovou vrstvu přepínače. Tento přístup má své výhody i nevýhody. Ve spojením s akcelerační knihovnou DPDK může být implementace datové vrstvy v uživatelském prostoru daleko rychlejší než datová vrstva implementovaná jádře OS (více v kapitole 3).

Open vSwitch ale dokáže v uživatelském prostoru fungovat kompletně i bez knihovny DPDK. Tento způsob použití je však považován za *experimentální* [34].

[13] RPC - Remote procedure call - technologie umožňující programu vykonat kód nacházející se na jiném místě, než je umístěn volající program (např. v síti).

[14] PKI - Public Key Infrastructure

Perzistentní databáze

Databáze uchovávající konfiguraci přepínače se ukládá do speciálních databázových souborů ve formátu JSON. Schéma databáze je přesně definováno [33]. Nad tímto databázovým souborem operuje `ovsdb-server`, který pro dotazy nad databází poskytuje protokol OVSDB [23]. Tento protokol umožňuje spravovat konfiguraci Open vSwitch vzdáleně (většinou z SDN řadiče). Umožňuje přidávat, mazat nebo upravovat virtuální rozhraní, vytvářet nová rozhraní atp. V určitých případech použití v SDN sítích tak ani není nutné komunikovat s Open vSwitch pomocí protokolu OpenFlow.

Databázový soubor musí být před spuštěním Open vSwitch předem vytvořen. Pokud tomu tak není, spuštění daemona `ovsdb-server` selže. Databázi lze vytvořit pomocí nástroje `ovsdb-tool`. Standardně se konfigurace přepínače ukládá do databázového souboru nacházejícího se v `/etc/openvswitch/conf.db`. Zde je důležité zmínit, že konfigurace přepínače Open vSwitch zůstává zachována mezi restarty systému.

Kapitola 3

Možnosti akcelerace přepínače Open vSwitch

V klasických operačních systémech jsou rámce přicházející na porty síťových karet zpracovávány asynchronně. Když síťová karta přijme nový rámec, uloží jej do své vyrovnávací paměti (cyklické fronty). Síťové karty v této frontě (popř. frontách) uchovávají více přijatých rámců. Po určité době je vyvoláno přerušení (IRQ[1]), které musí jádro operačního systému zpracovat. Při každém takovém zpracování je nutné pozastavit činnost procesoru, vyvolat tzv. proceduru obsluhy přerušení (ISR[2]), a poté obnovit činnost procesoru ve stejném místě, kde bylo přerušení vyvoláno. Celý tento proces se nazývá *změna kontextu* a jedná se o výpočetně náročnou operaci [13][2].

Obsluha přerušení je většinou přímou součástí ovladače daného I/O zařízení. Funkce, kterou musí obsluha přerušení vykonat, může být různá. V případě síťových karet jde o zkopírování rámců z fronty síťové karty do operační paměti.

Obsluha každého z přerušení vyžaduje provést nemalý počet taktů procesoru [13]. Pokud bude muset takový systém přijmout velký počet rámců, může se jednoduše stát, že se fronty síťových karet velmi rychle zaplní a začnou generovat vysoký počet přerušení. Jádro operačního systému bude přerušeními zahlceno a nebude je stíhat obsluhovat. Propustnost takového systému pak rychle klesá. Tyto problémy se netýkají pouze karet fyzických, ale i virtualizovaných rozhraní [5][40].

Dnešní operační systémy nejsou navrženy na dlouhotrvající příjímání velkého počtu malých rámců. Existují ale případy použití, kdy se zpracování velkého množství malých rámců nevyhneme a zároveň bychom chtěli mít přítomen i operační systém, který by určitým způsobem dále zajišťoval přístup ke zdrojům.

Proto vznikají alternativní technologie, které se výše zmíněné problémy snaží řešit. Tyto technologie se většinou snaží vyhýbat asynchronnímu zpracování přerušení, popř. se vyhýbají zpracování dat v síťové vrstvě jádra [5].

[1] IRQ - Interrupt request
[2] ISR - Interrupt service routine

Zajímavé je, že se tyto technologie částečně vrací zpět do dřívějších dob, kdy operační systémy pro čtení dat z I/O zařízení využívaly aktivní čekání ve smyčce (tzv. polling[3]). Polling může být v některých případech velmi efektivní - především z pohledu nízké latence.

Příkladem knihoven pro vysokorychlostní zpracování je Netmap a DPDK. Obě z technologií využívají pollingu namísto asynchronního zpracování pomocí přerušení [27][26][17]. Knihovnou DPDK se tato práce bude zabývat především.

3.1 Knihovna DPDK a její architektura

Data Plane Development Kit (zkráceně DPDK) je soubor knihoven a ovladačů pro vysokorychlostní zpracování paketů. Knihovna je kompletně napsána v jazyce C a její kód licencován pod licencí BSD. Cílem knihovny DPDK je implementovat datovou vrstvu síťového zařízení v software (běžící přímo na CPU). Ta má svým výkonem konkurovat dnešním velmi drahým síťovým jednoúčelovým hardwarovým zařízením, která datovou vrstvu implementují pomocí ASIC technologií.

DPDK poskytuje programové rozhraní pro procesory rodiny Intel x86. Podporuje procesory Intel Atom, Intel Xeon, ale i jiné procesorové architektury jako POWER8. Pro dosažení maximálního výkonu nicméně oficiální dokumentace DPDK doporučuje použít procesory Intel Xeon s mikroarchitekturou Ivy Bridge, Haswell nebo novější [17].

Celá technologie využívá přímého přístupu k odesílacím (TX) a přijímacím (RX) frontám síťových karet. Nevyužívá se zde princip přerušení (s výjimkou přerušení indikujícího změnu stavu linky). Obdržení nově přijatých dat se provádí pomocí *pollingu* nad RX frontami. Polling je prováděn ovladačem síťové karty, tzv. PMD (více lze najít v sekci 3.1.2). Smyslem tohoto přístupu je co nejrychleji zkopírovat data z RX front síťové karty do interní cyklické fronty v paměti (3.1.2). Polling tak zaručuje velmi nízkou latenci na úkor *většího procesorového času* a *vyšší celkové spotřeby energie*.

Důležité je zmínit, že DPDK není implementace síťové protokolové vrstvy (z angl. protocol stack). Neposkytuje funkce vyšších vstev jako přepínání, směrování, překlad adres, IPsec nebo firewall. Zmiňované funkce je nutné s pomocí knihovny implementovat v uživatelském prostoru. Projekty poskytující zmíněné služby postupně vznikají [8]. Již dnes ji integrují projekty jako FD.io [38], realistický generátor provozu T-Rex vyvíjený společností Cisco [30] nebo již zmíněný virtuální přepínač Open vSwitch (viz 2.2.4).

3.1.1 Požadavky pro bezproblémovou funkčnost knihovny

Knihovna DPDK dokáže fungovat na většině moderních x86_64 architekturách. A to nejen na fyzických (bare-metal), ale i na virtuálních. Není problém vytvořit DPDK aplikaci běžící ve virtuálním hostovi, která bude pro komunikaci využívat paravirtualizované ovladače (např. ovladač virtio[4]).

Pro maximální výkon, kvůli kterému byla knihovna vytvořena především, musí architektury disponovat určitými technologiemi. V následujících částech se podíváme na ty nejdůležitější [17].

[3] polling - neboli „aktivní čekání ve smyčce“, se využíval před příchodem přerušení. Když procesor potřeboval zjistit stav I/O zařízení, musel se ho explicitně dotázat. Problém tohoto přístupu byl ten, že procesor je většinou o mnoho rychlejší než periferní zařízení. Často se tak stávalo, že procesor se na stav zařízení dotázal mnohokrát, i když I/O operace stále nebyla připravena. Dostával se tak do stavu tzv. pasivního čekání.

[4] http://dpdk.org/doc/guides/nics/virtio.html

Základní softwarové požadavky

Knihovna DPDK je určena především pro běh na operačním systému GNU/Linux. Existuje i verze pro systém FreeBSD. Ta má ale v porovnání s verzí pro GNU/Linux velmi omezené možnosti. Tato práce se bude zabývat především DPDK knihovnou, která je určená pro systém GNU/Linux.

Pro správný běh knihovny musejí být splněné určité minimální požadavky na software. Jedním z nich je požadavek na minimální verzi knihovny glibc. Její verze musí být alespoň `2.7`. Další důležitý požadavek je na linuxové jádro. Jádro musí být minimálně verze `2.6.34` a musí být přeloženo s podporou pro HugeTLBFS[5], Userspace I/O[6] a volbou `CONFIG_PROC_PAGE_MONITOR`.

Podpora síťových karet

Seznam podporovaných síťových karet lze nalézt na oficiálních webových stránkách knihovny. Mezi aktuálně podporované patří především karty od společnosti Intel. DPDK ale podporuje i karty jiných výrobců - Broadcom, Chelsio, Cisco, Emulex, Mellanox, QLogic nebo karty vyvíjené společností Cesnet - COMBO-80G a COMBO-100G. Dále jsou podporovány i karty paravirtualizované - virtio-net (QEMU), xenvirt (XEN), vmxnet3 (VMware ESXi). Pokud chceme mít jistotu, že karty budou s DPDK spolupracovat tak jak mají, je důležité, aby byl jejich firmware zaktualizován na poslední verzi [17].

Jádra procesoru a jejich izolace

V současné době se na trhu čím dál méně setkáváme s jednojádrovými procesory. Je to především z toho důvodu, že výrobci procesorů chtěli stále zvyšovat výkon pomocí zvyšování počtu tranzistorů. Počet tranzistorů, které mohou být umístěny na čipu, se při zachování stejné ceny zhruba každé dva roky zdvojnásobí (tzv. *Mooreův zákon*). Ačkoliv tranzistorů na čipu stále přibývá, zvyšování frekvence procesorů se skoro zastavilo. Výrobci proto přešli na koncept procesorů vícejádrových. Vícejádrové procesory jsou mikroprocesory, které v jednom pouzdře nebo na jednom čipu integrují dvě a více jader [11].

Pro optimální funkci DPDK knihovny je doporučováno vyhradit různá procesorová jádra k různým funkcím. Jádra tak mohou tuto funkci obstarávat na plný výkon aniž by byla jejich činnost narušována jinými procesy. Prvně je nutné provést tzv. *izolaci jader*, které je schopna zajistit, že na dané množině jader (většinou specifikované pomocí masky) nebudou plánovačem plánovány žádné procesy. V případě DPDK bude část těchto izolovaných jader vyhrazena pouze pro běh PMD ovladačů provádějících polling nad síťovou kartou (více v sekci 3.1.2). Další část izolovaných procesorových jader je vyhrazena pro samotný běh DPDK aplikace. Tou může být právě například přepínač Open vSwitch [17].

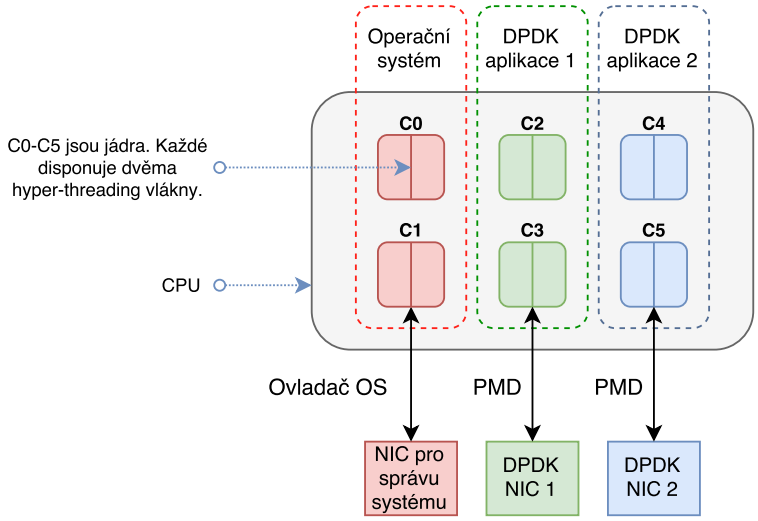
Příklad mapování izolovaných jader procesoru na jednotlivé aplikace DPDK lze vidět na obrázku 3.1.

K izolaci jader procesoru lze na operačním systému GNU/Linux využít parametru jádra `isolcpus`[7].

[5] HugeTLBFS - virtuální filesystém běžící v RAM poskytující souborové rozhraní pro práci s pamětí - konkrétně s velkými stránkami paměti. Více informací na https://lwn.net/Articles/375096 nebo v sekci 3.1.1.

[6] https://lwn.net/Articles/232575

[7] Více lze najít v dokumentaci jádra na https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt



Obrázek 3.1: Využití jader procesoru pro různé DPDK aplikace.

Lokalita dat

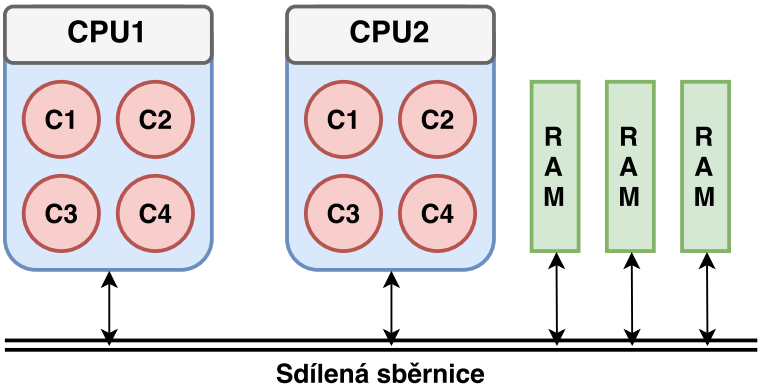
Z důvodu zvyšování počtu jader procesorů a stálého stoupání výkonnosti se čím dál více dostává do popředí důležitost architektury operační paměti. První vícejádrové procesory do paměti přistupovaly přes sdílenou sběrnici. Všechna jádra tak sdílela jednu *společnou paměť*. Tato architektura se nazývá UMA[8]. S přibýváním počtu jader ale vzniká problém, kdy se sdílená sběrnice stává úzkým místem pro přístup do paměti. Jako řešení vznikly paměťové architektury typu NUMA[9]. V této architektuře jsou jednotlivá jádra seskupena do jednoho celku zvaného NUMA uzel. Každý takový uzel má přiřazenu svou lokální paměť. Paměť, která je součástí jiného NUMA uzlu, se nazývá *vzdálená*. Přístup do vzdálené paměti zabere více času než přístup do paměti lokální [11].

Na obrázku 3.2 lze vidět příklad architektury UMA s dvěma procesory (každý o čtyřech jádrech) propojenými sdílenou sběrnicí ke společné paměti.

Obrázek 3.3 znázorňuje příklad architektury NUMA. Zde každý z procesorů disponuje svou vlastní lokální pamětí. Ke vzdálené paměti každý z procesorů může přistoupit přes propojení znázorněná dvojitou čarou. Propojení může být realizováno například pomocí QPI[10].

Paměť (v případě DPDK především velké stránky - viz sekce 3.1.1), která je používána aplikací běžící pod DPDK, by měla být alokována na stejném NUMA uzlu, na kterém aplikace běží. Pokud je procesor vybaven PCI-Express řadičem, měla by alokace proběhnout na stejném NUMA uzlu, ke kterému je připojeno síťové zařízení využívané DPDK aplikací. Je to z toho důvodu, aby nedocházelo ke kopírování dat mezi síťovou kartou a vzdálenou pamětí.

[8] UMA - Uniform Memory Access
[9] NUMA - Non-uniform Memory Access
[10] QPI - Intel QuickPath Interconnect



Obrázek 3.2: Příklad architektury UMA.

IOMMU

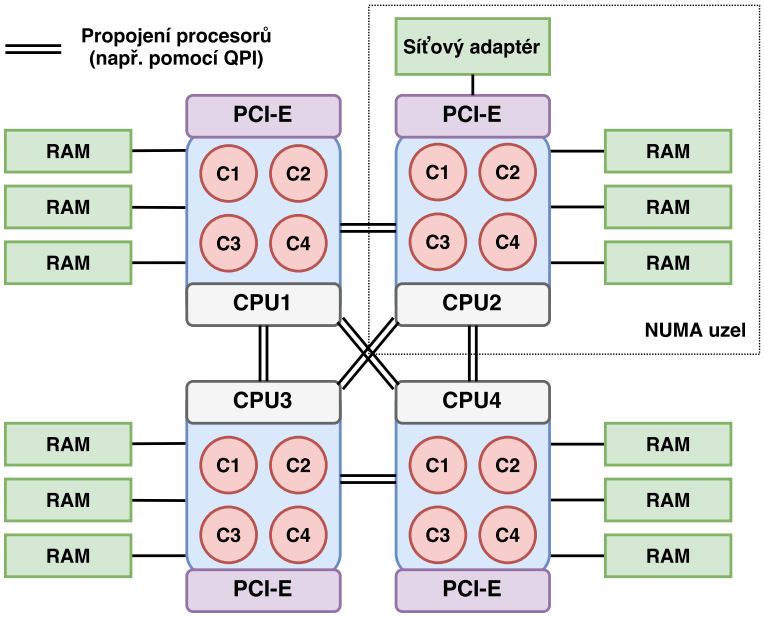
Součástí dnešních systémů je jednotka zvaná MMU (Memory Management Unit). Tato jednotka je většinou přímou součástí procesoru, ale na některých architekturách může existovat i samostatně. MMU má za úkol *překlad virtuálních adres stránek na adresy fyzické*, ochranu paměti a arbitraci paměťové sběrnice.

Na některých novějších architekturách existuje jednotka zvaná IOMMU (Input-Output Memory Management Unit). Ta propojuje vstupně/výstupní sběrnici a hlavní operační paměť. Funguje velmi podobně jako MMU. Pouze nepřekládá virtuální adresy procesoru, ale adresy používané vstupně/výstupními zařízeními (zvané vstupně/výstupní adresy, popř. adresy zařízení). Vstupně/výstupní zařízení tak mají přístup pouze k regionům paměti (tzv. doménám), které jim procesor pomocí řadiče IOMMU povolí. Je tak zajištěna *ochrana před neoprávněným přístupem do paměti*. Bez IOMMU by přímý přístup do paměti stále fungoval, vstupně/výstupní zařízení by ale měla přístup do jakéhokoliv regionu paměti. Mohly by tak, ať už cíleně nebo ne, porušit paměť jádra operačního systému. Jádro by muselo DMA[11] operace explicitně hlídat [14].

Na obrázku 3.4 lze vidět porovnání jednotek MMU a IOMMU v rámci přístupu k hlavní fyzické paměti.

Jednotka IOMMU má své využití hlavně u virtualizovaných prostředí, ve kterých je nutné nedůvěryhodným virtuálním hostům poskytnout přístup k fyzickému zařízení na systému (tzv. device passthrough). Zároveň ale virtuální host nesmí přes DMA periferního zařízení přistupovat do paměti jiných virtuálních hostů, samotného hypervizoru, popř. operačního systému, na kterém hypervizor běží. Pokud systém jednotkou IOMMU nedisponuje, musí hypervizor všechny přístupy do paměti hlídat (např. pomocí emulace). Tento přístup představuje nemalou režii a tedy i potenciální problém s výkonem. S využitím jednotky IOMMU lze vyhradit části fyzické paměti (domény), do které bude umožněno kopírovat data z periferního zařízení (s využitím DMA) bez jakéhokoliv zásahu hypervizoru. Hypervizor zasáhne pouze v případě, když host přistoupí na adresu nacházející se mimo povolenou doménu [41].

[11] DMA (Direct Memory Access) - umožňuje vstupně/výstupním zařízením přímý přístup do operační paměti. Data tak neprocházejí skrz procesor, ten pouze programuje řadič DMA a přenos zahájí.



Obrázek 3.3: Příklad architektury NUMA.

Jelikož knihovna DPDK běží v uživatelském prostoru a se síťovým zařízením pracuje pomocí nízkoúrovňového rozhraní jádra (viz sekce 3.1.2), je použití jednotky IOMMU více než žádoucí. Jádro nemusí kontrolovat každé kopírování ze síťové karty do paměti jednoduše přenechá na jednotce IOMMU.

IOMMU dnes podporuje většina nejnovějších základních desek a procesorů značek AMD i Intel[12]. U procesorů AMD se výše zmíněná virtualizace nazývá také *AMD-Vi*, u procesorů značek Intel se jedná o *VT-d* (Intel Virtualization Technology for Directed I/O).

Podpora pro jednotku IOMMU nemusí být v linuxovém jádře standardně zapnuta. Její aktivace se provádí pomocí parametru `intel_iommu=on` na procesorech Intel, `amd_iommu=on` na procesorech od firmy AMD.

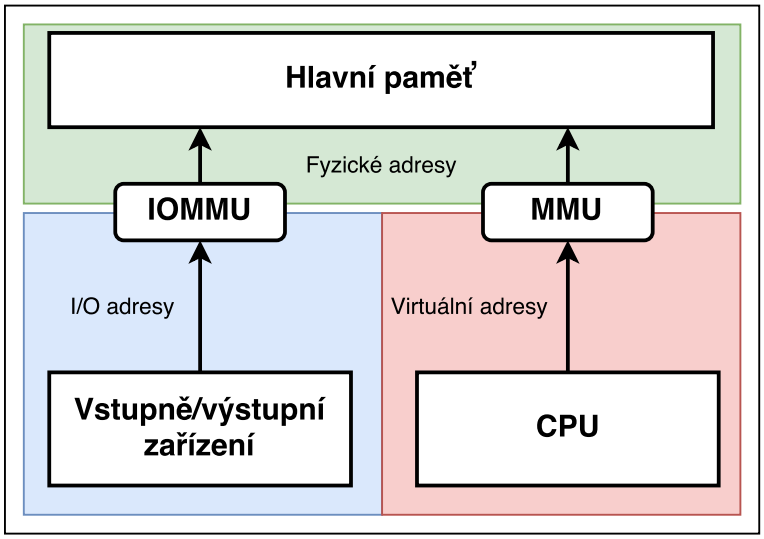
Podpora velkých stránek

Podpora velkých stránek (tzv. hugepages (Linux), Super pages (BSD) nebo Large Pages (Windows)) je pro správný běh DPDK velmi důležitá. DPDK knihovna totiž velké stránky využívá k alokaci velkých paměťových bloků používaných k ukládání síťových paketů. Důvodem pro použití velkých stránek je zvýšení výkonu. Při jejich použití je zapotřebí menší počet stránek než kdyby byly použity stránky standardní (většinou 4 KiB velké). Tím se snižuje zatížení cache TLB[13], nastává menší počet výpadků stránek (tzv. TLB miss) a díky tomu se snižuje i čas potřebný k přeložení virtuální adresy stránky na fyzickou [17].

Velké stránky potřebují mít podporu na procesoru, což není samozřejmostí. Většina dnešních architektur už ale stránky větší jak 4 KiB podporuje. Například na architektuře

[12]Seznam HW podporujícího IOMMU lze nalézt na https://en.wikipedia.org/wiki/List_of_IOMMU-supporting_hardware

[13]TLB (Translation Lookaside Buffer) - velmi rychlá cache, která je součástí jednotky správy paměti (již zmíněné MMU 3.1.1). TLB má překlady adres prováděné jednotkou MMU urychlit.



Obrázek 3.4: Porovnání jednotek MMU a IOMMU.

x86_64 lze očekávat podporu velkých stránek o velikost 2 MiB, popř. 1 GiB. Pro 64-bitové aplikace je doporučeno používat velké stránky o velikosti 1 GiB.

Rezervace velkých stránek by se měla provádět už při startu systému, popř. co nejdříve po startu. Důvodem je předejít fragmentaci fyzické paměti. V prostředí GNU/Linux lze stránky rezervovat už při startu pomocí několika parametrů jádra. Parametr `hugepages=N` říká jádru kolik stránek má rezervovat[14]. Druhý parametr `hugepagesz=<velikost>` určuje velikost jednotlivých velkých stránek. Tato velikost je závislá na platformě. Posledním parametrem ovlivňujícím chování velkých stránek je `default_hugepagesz=<velikost>`. Ten určuje velikost alokovaných stránek [32].

Pro rezervaci čtyř stránek o velikosti 1 GiB by se použily parametry z ukázky 3.1.

```
default_hugepagesz=1G hugepagesz=1G hugepages=4
```

Kód 3.1: Příklad rezervace velkých stránek při startu systému.

Velké stránky o velikosti 2 MiB lze rezervovat i po startu systému pomocí zapsání počtu požadovaných stránek do speciálního souboru v sysfs[15]. Příklad 3.2 ukazuje rezervaci čtyř velkých stránek o velikosti 2 MiB na každém ze dvou NUMA uzlů [12].

```
echo 4 >/sys/devices/system/node/node0/hugepages/hugepages-2048kB/nr_hugepages
echo 4 >/sys/devices/system/node/node1/hugepages/hugepages-2048kB/nr_hugepages
```

Kód 3.2: Příklad rezervace velkých stránek po startu systému.

Poté co jsou stránky úspěšně zarezervovány, musíme k nim umožnit DPDK aplikaci přístup. Ten zajistíme pomocí již zmíněného HugeTLBFS. Příklad 3.3 ukazuje připojení tohoto speciálního souborového systému.

[14] V případě NUMA systémů jsou velké stránky rovnoměrně rozděleny mezi jednotlivé NUMA uzly (za předpokladu, že disponují dostatkem volné paměti).

[15] https://www.kernel.org/doc/Documentation/filesystems/sysfs.txt

```
mkdir /mnt/huge
mount -t hugetlbfs -o "pagesize=1G" nodev /mnt/huge
```

Kód 3.3: Příklad připojení filesystému HugeTLBFS.

3.1.2 Architektura DPDK

Celé DPDK se skládá z několika komponent (především knihoven), které spolu navzájem spolupracují. Ty nejdůležitější jsou zde rozebrány [17][8].

Environment Abstraction Layer

První důležitou komponentou je *Environment Abstraction Layer* (neboli EAL). EAL poskytuje obecné rozhraní abstrahující přístup k nízkoúrovňovým zdrojům dané architektury. Zajišťuje také inicializaci těchto zdrojů.

EAL poskytuje následující služby:

- Načtení a spuštění DPDK - knihovna DPDK a aplikace, která využívá její služeb, musí být inicializována.
- Konfiguraci afinity[16] CPU jader a jejich inicializaci.
- Rezervaci nebo uvolnění fyzické paměti pomocí funkce `mmap()` v HugeTLBFS.
- Zajištění abstrakce k adresám PCI.
- Speciální funkce jako zámky nebo atomické čítače, které nejsou součástí knihovny *libc*.
- Správu přerušení. Konkrétně poskytuje rozhraní, které umožňuje pro určitá přerušení zaregistrovat tzv. callback[17] funkci.
- Funkce pro trasování a ladění.

Knihovna implementující strukturu cyklické fronty

Jednou z nejdůležitějších struktur v DPDK používanou především pro komunikaci mezi DPDK aplikacemi, alokaci paměti pomocí knihovny Mempool (viz 3.1.2) nebo pro komunikaci mezi procesorovými jádry, je *cyklická fronta*. Její implementace je součástí knihovny `librte_ring`. Cyklická fronta má na rozdíl od klasického seznamu následující vlastnosti:

- FIFO (first in, first out)
- Po inicializaci je její maximální velikost pevná a nelze měnit.
- Pro její implementaci nejsou použity synchronizační zámky. Synchronizace je zajištěna pomocí atomických operací.
- Podpora více než jednoho producenta nebo konzumenta.

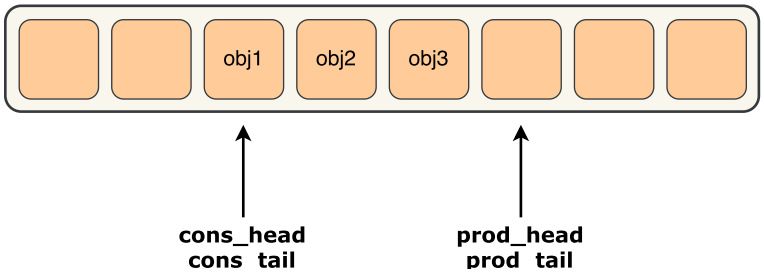
[16] Afinita CPU (popř. také tzv. CPU pinning) umožnuje specifikovat procesu CPU jádro (popř. množinu jader), na kterých je mu umožněno běžet. V podstatě se jedná o úpravu planovacího algoritmu. Afinita je většinou specifikována pomocí bitové masky.

[17] Callback - https://en.wikipedia.org/wiki/Callback_(computer_programming)

- Umožňuje přidat/odebrat více než jeden objekt do/z fronty najednou (v rámci jedné operace). Tato operace je nazývána jako tzv. *bulk enqueue/dequeue*.

Výhody této struktury, oproti klasické frontě implementované pomocí lineárního spojového seznamu, spočívají především v rychlosti a podpoře pro dávkové operace. Zároveň není nijak náročná na implementaci.

Příklad struktury o kapacitě osm prvků lze vidět na obrázku 3.5. Jsou zde vidět čtyři ukazatele. První dva (`cons_head` a `cons_tail`) jsou používány při odebírání prvků z fronty konzumentem. Zbylé dva (`prod_head` a `prod_tail`) jsou využívány producentem při přidávání jednoho nebo více prvků do fronty.



Obrázek 3.5: Příklad struktury cyklické fronty.

Více informací o této struktuře a její implementaci lze najít v programové dokumentaci [17].

Knihovna Mempool

Paměť tvořená stránkami (viz 3.1.1) je na počátku inicializována vrstvou EAL (viz 3.1.2). Knihovna Mempool (`librte_mempool`) zajišťuje práci s pamětí na vyšší úrovni. Paměť je organizována do částí o pevné velikosti, do tzv. *poolů*. V těchto poolech jsou uchovávány objekty. Základní používanou strukturou pro implementaci poolu je již zmíněná cyklická fronta (viz 3.1.2). Mempool také řeší zarovnání objektů v paměti (tzv. padding). V závislosti na hardwarové konfiguraci operační paměti mempool umožňuje přidávat mezi objekty prázdná místa. Cílem je zajistit, aby každý z objektů začínal v paměti typu DIMM[18] v jiném kanálu a ranku[19]. Díky tomu lze využít plného potenciálu paměti a dosáhnout tak větší propustnosti.

Mempool poskytuje ještě další volitelné služby jako například cache. Tuto cache si mempool udržuje pro každé jádro CPU, na kterém DPDK běží. Cache slouží k tomu, aby se snížil počet přístupů do hlavního paměťového poolu - především u aplikací běžících na více jádrech. Přístupy do paměťového poolu je totiž nutné synchronizovat a časté použití synchronizačních zámků by se na výkonu projevilo velmi negativně. Takto může každé z jader přistupovat do své vlastní lokální cache a s ní pracovat. Přístup do hlavního paměťového poolu je potřeba pouze v případě, že se lokální cache zaplní nebo je potřeba naplnit objekty [17].

[18] https://en.wikipedia.org/wiki/DIMM
[19] https://en.wikipedia.org/wiki/Memory_geometry

Knihovna Mbuf (Message buffer)

Knihovna DPDK potřebuje manipulovat s příchozími popř. odchozími síťovými rámci. K tomu slouží datová struktura zvaná *message buffer*, zkráceně také `mbuf`. Struktura `mbuf` v sobě může přenášet buď kontrolní zprávy (tzv. *generic control buffers*) a nebo klasické síťové rámce (tzv. *network packet buffers*). Knihovna Mbuf poskytuje rozhraní pro práci s těmito strukturami. Umožňuje jejich vytvoření, naplnění daty, řetězení, zrušení nebo přesouvání. Struktury `mbuf` jsou vytvořeny na počátku startu DPDK aplikace a jsou uloženy v paměťových poolech. Proto knihovna Mbuf interně využívá služeb knihovny Mempool (viz 3.1.2).

Ovladače síťových karet (Poll Mode Drivers)

Pro komunikaci s řadiči síťových karet jsou zapotřebí ovladače. V klasickém operačním systému jsou tyto ovladače součástí jádra, popř. jeho modulů. DPDK ale běží v uživatelském prostoru a jádro je zde využíváno pouze pro nízkoúrovňovou komunikaci s I/O zařízením. Proto je nutné, aby ovladače byly přímou součástí DPDK. Knihovna DPDK obsahuje ovladače pro vybrané 1GbE, 10GbE, 40GbE a paravirtualizované (virtio) ethernetové adaptéry. Tyto ovladače se v terminologii DPDK nazývají tzv. *Poll Mode Drivers* (zkráceně PMD).

Aby síťová karta mohla fungovat pod DPDK, musí být jádro informováno o tom, že má pro komunikaci s kartou používat ovladač jiný než ovladač jaderný - v případě DPDK nízkoúrovňový. Tomuto procesu se říká tzv. *binding* a *unbinding* zařízení. Linuxové jádro od verze 2.6.13-rc3 binding a unbinding podporuje. Síťovou kartu lze od jaderného ovladače oddělit a přidělit ji pod správu ovladači jinému pomocí zápisu do speciálního souboru v sysfs [15]. Následující příkazy (3.4) odeberou síťovou kartu s identifikátorem sběrnice „`0000:07:00.0`“ ze správy ovladače `igb` a přiřadí ji pod kontrolu ovladači `vfio-pci`.

```
echo -n "0000:07:00.0" > /sys/bus/pci/drivers/igb/unbind
echo -n "0000:07:00.0" > /sys/bus/pci/drivers/vfio-pci/bind
```

Kód 3.4: Příklad unbindingu a bindingu síťové karty.

Jak lze vidět, změna ovladače karty je jednoduchá, ale není intuitivní. Proto DPDK obsahuje nástroj, který binding a unbinding zjednodušuje. Skript se jmenuje `dpdk-devbind`. Ukázku jeho použití lze vidět v příkladě kódu 3.5.

```
dpdk-devbind --bind=vfio-pci "0000:07:00.0"
```

Kód 3.5: Binding pomocí skriptu dpdk-devbind.

Aktuální stav síťových zařízení a jejich ovladačů lze zobrazit pomocí parametru `--status`.

Knihovna DPDK ve verzi v16.11 podporuje tři nízkoúrovňové ovladače pro práci se síťovou kartou. Jsou jimi `vfio-pci`, `uio_pci_generic` a `igb_uio`.

Jakmile lze s kartou komunikovat pomocí nízkoúrovňového ovladače, nastupuje na řadu ovladač PMD. Ten má za úkol inicializovat řadič síťové karty (globální reset do známého stavu) a nakonfigurovat její RX a TX fronty. Následně PMD přistupuje k těmto frontám pomocí jejich deskriptorů přímo - bez použití přerušení. Díky tomu může velmi rychle příchozí data přijmout, zpracovat a předat cílové aplikaci.

Kernel NIC Interface (KNI)

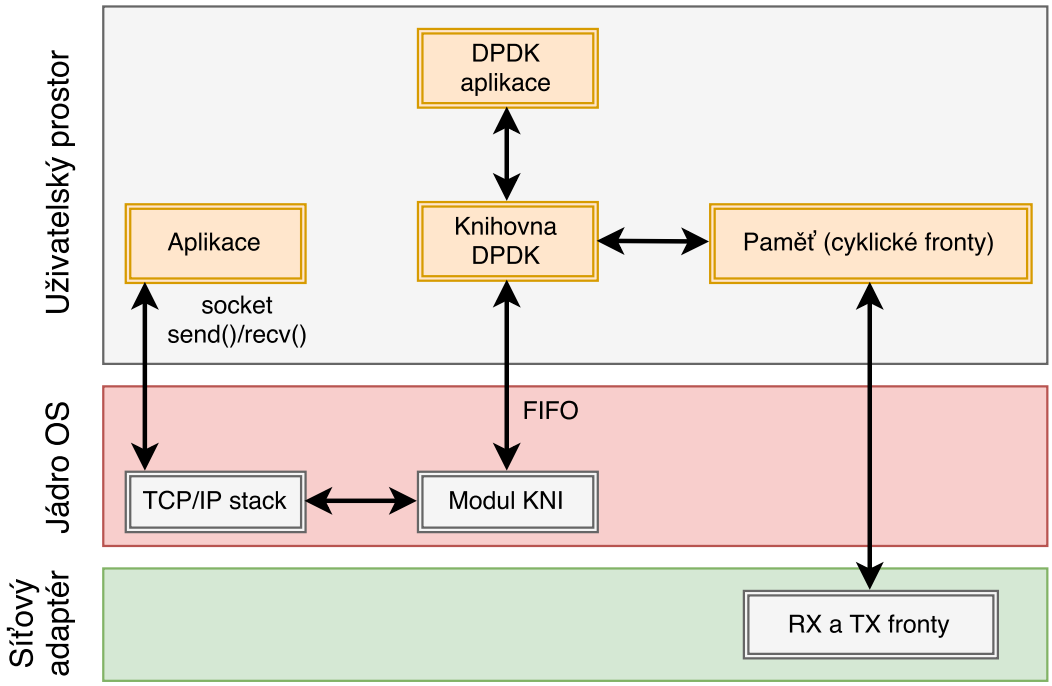
Jak již bylo zmíněno v sekci 3.1, DPDK pro práci se síťovou kartou využívá pouze nízkoúrovňové operace jádra operačního systému. Ovladač karty (PMD) běží v uživatelském prostoru. Následkem toho je, že v systému pro síťové zařízení není vytvořeno virtuální síťové rozhraní (standardně ho vytváří ovladač karty běžící v jádře). Pro práci s kartou v rámci OS tedy nelze používat standardní programy jako utility z balíku `iproute2`[20], `ethtool`[21], `tcpdump`[22] nebo firewall (`iptables` či novější `nftables`).

Nastávají situace, kdy je nutné mít možnost propojit aplikaci běžící pod DPDK se síťovou vrstvou jádra OS. Jednou z možností by bylo použít jaderný modul TUN/TAP[23]. Ten ale v případě DPDK není dostatečně rychlý, protože zde dochází k častému volání služeb jádra a ke kopírování z/do uživatelského prostoru pomocí jaderných funkcí `copy_from_user()` a `copy_to_user()` [17].

Vývojáři knihovny proto vytvořili jaderný modul zvaný KNI. Tento modul není součástí oficiálního vydání linuxového jádra. Je jej nutné explicitně doinstalovat, popř. přeložit ze zdrojových kódů a zavést. Pomocí tohoto modulu může DPDK aplikace pomocí volání jádra `ioctl()` dynamicky vytvářet nová virtuální zařízení. Tato zařízení jsou z pohledu OS plnohodnotná. Každé z nich má svou MAC adresu, lze jim nastavit IPv4 adresu, IPv6 adresu, MTU a mnoho dalšího. Lze je konfigurovat výše zmíněnými nástroji, popř. lze nad nimi provádět filtrování pomocí firewallu. Vše co do virtuálního rozhraní vstoupí, je zpracováno jadernou síťovou vrstvou a předáno modulu KNI. Zde dochází k naplnění struktury `mbuf` (viz 3.1.2) daty z jaderné struktury `sk_buff`[24]. Struktura `sk_buff` je následně uvolněna a `mbuf` je předána RX frontě. Nad touto RX frontou už provádí knihovna DPDK polling stejně jako by tomu bylo u klasického síťového rozhraní pracujícího pod DPDK. Z RX fronty je `mbuf` odebrán, zpracován ovladačem PMD a uvolněn. V případě přijímání dat virtuálním rozhraním typu KNI probíhá vše obráceně.

Na obrázku 3.6 lze vidět znázornění komunikace mezi DPDK, modulem KNI a uživatelskou aplikací.

[20] https://git.kernel.org/cgit/linux/kernel/git/shemminger/iproute2.git
[21] https://www.kernel.org/pub/software/network/ethtool
[22] http://www.tcpdump.org
[23] https://www.kernel.org/doc/Documentation/networking/tuntap.txt
[24] `sk_buff` - neboli *socket buffer* je nejzákladnější strukturou síťové vrstvy linuxového jádra. Každý přijatý nebo odeslaný paket je popsán právě touto strukturou [32].



Obrázek 3.6: Znázornění modulu KNI v rámci systému.

Kapitola 4

Metodologie testování

Metodologie testování výkonnosti je z velké části založena na doporučeních popsaných v dokumentu zabývajícím se testováním virtuálních přepínačů [29]. Tento dokument byl vytvořen v rámci iniciativy OPNFV (Open Platform for NFV), která se zabývá vytvořením otevřené referenční platformy skládající se ze síťových virtuálních funkcí (tzv. VNF[1]). U těchto virtuálních funkcí se projekt OPNFV snaží zajistit jejich vzájemnou konzistenci a výkonnost. Nutno zdůraznit, že za celým projektem OPNFV stojí mnoho velkých společností jako AT&T, Red Hat, SUSE, IBM, Cisco, Juniper, VMware, Intel a další [36].

4.1 RFC2544

Samotné měření výkonnosti probíhá na základě RFC2544 *(Benchmarking Methodology for Network Interconnect Devices)* [20] a RFC2889 *(Benchmarking Methodology for LAN Switching Devices)* [18]. Následující odstavce využívají terminologii popsanou v RFC1242 *Benchmarking Terminology for Network Interconnection Devices* [3].

Měření se vždy účastní dva systémy. Na prvním z nich je nakonfigurován generátor síťového provozu (z angl. traffic generator). Na druhé straně je testovaný systém nazývaný DUT (z angl. Device Under Test). Na tomto systému je nakonfigurována vždy jedna z topologií Open vSwitch (viz 4.2).

V produkčním prostředí by pravděpodobně na DUT běžela virtuální síťová funkce (např. směrovač nebo firewall). V případě implementovaných testů provádí DUT funkci přepínače. Přijatá data jsou pouze přeposílána dále (zpracování vyšších vrstev než L2 se neprovádí). Testy tedy měří propustnost celého systému na vrstvě L2.

4.1.1 Měření propustnosti

Měření propustnosti je doporučeno provádět následujícím způsobem. Generátor provozu odešle určitý počet rámců určitou rychlostí[2] na DUT. Zároveň sleduje počet rámců, které mu byly odeslány zpět z DUT. Přijaté rámce musí být verifikovány - mezi přijaté se počítají pouze ty, které odpovídají rámcům odeslaným. RFC2544 doporučuje verifikaci provádět na základě velikosti přijatých rámců, která by měla odpovídat velikosti rámců odeslaných, popř. pomocí číslování - využitím sekvenčních čísel. Do počtu přijatých rámců by se neměly

[1] VNF (Virtual Network Function) - Virtuální síťová funkce představuje službu běžící ve virtualizovaném prostředí, která zajišťuje nějakou ze síťových funkcí - například přepínání, směrování, firewall, apod.

[2] Na počátku je tato rychlost určena dle maximální teoretické propustnosti přenosového média. Pro 10Gbit linku a ethernetové rámce o velikosti 64 B je jejich maximální rychlost odesílání 14880952 pps.

započítávat například rámce pro keep-alive apod. [20]. Pokud je počet přijatých rámců menší než počet odeslaných, je rychlost odesílání snížena a celý test se opakuje. Propustnost je pak taková maximální rychlost, při které se počet odeslaných rámců rovná počtu přijatých rámců z DUT (tedy ztrátovost rámců je nulová).

Jelikož reálný síťový provoz probíhá většinou v obou směrech, jsou i data odesílaná generátorem provozu na DUT vždy generována *v obou směrech*. Toto doporučení přímo zmiňuje RFC2544 [20].

Velikosti rámců použitých při testování

RFC2544 doporučuje měření provádět pro větší počet velikostí rámců. Do testování by měly být zařazeny rámce nejmenší[3] i největší možné velikosti daného testovaného protokolu. V případě ethernetu je doporučeno testovat rámce o velikostech 64, 128, 256, 512, 1024, 1280 a 1518 bajtů [20].

4.1.2 Formát výstupu

Dle doporučení RFC2544 by měly být výsledky znázorněny formou grafu. Hodnotami na ose X by měly být velikosti testovaných rámců. Hodnoty na ose Y by měly znázorňovat propustnost vyjádřenou v počtu přenesených rámců za sekundu, popř. v bitech/bajtech za sekundu. V grafu by pro každou velikost rámce měla být znázorněna jak maximální teoretická propustnost média, tak i propustnost naměřená v rámci výkonnostního testu.

4.2 Testované konfigurace

Pro účely měření byly zvoleny dvě základní topologie. Jednodušší topologie zvaná P2P (z angl. physical-to-physical) a jedna složitější PVP (z angl. physical-virtual-physical).

Kromě dvou zmíněných topologií existují ještě složitější. Jejich popis, konfigurace a měření by bohužel bylo nad rámec této práce. Příkladem takové topologie je PVVP. Ta disponuje dvěma virtuálními hosty (běžícími buď v rámci jednoho nebo dvou NUMA uzlů), ve kterých dochází k přeposílání rámců.

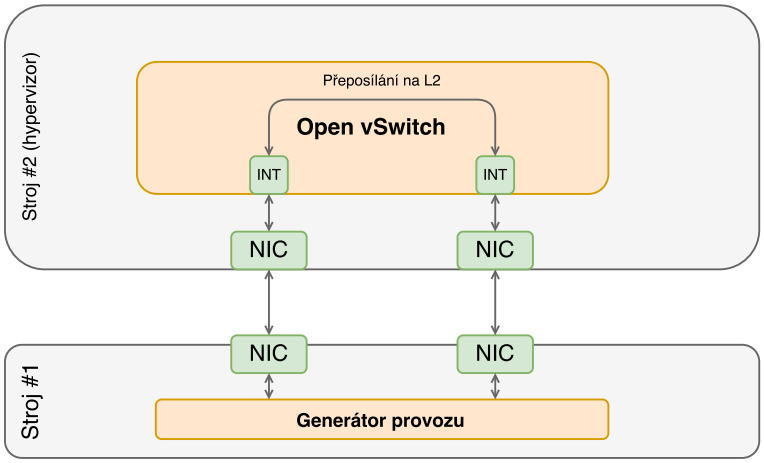
4.2.1 Topologie P2P

Jedná se o jednu z nejzákladnějších topologií, ve které jsou fyzická rozhraní síťové karty přímo propojené s virtuálním přepínačem. K přeposílání rámců dochází pouze na úrovni přepínače Open vSwitch. Komunikace se nijak neúčastní virtuální host. Síťové zapojení topologie P2P lze vidět na obrázku 4.1.

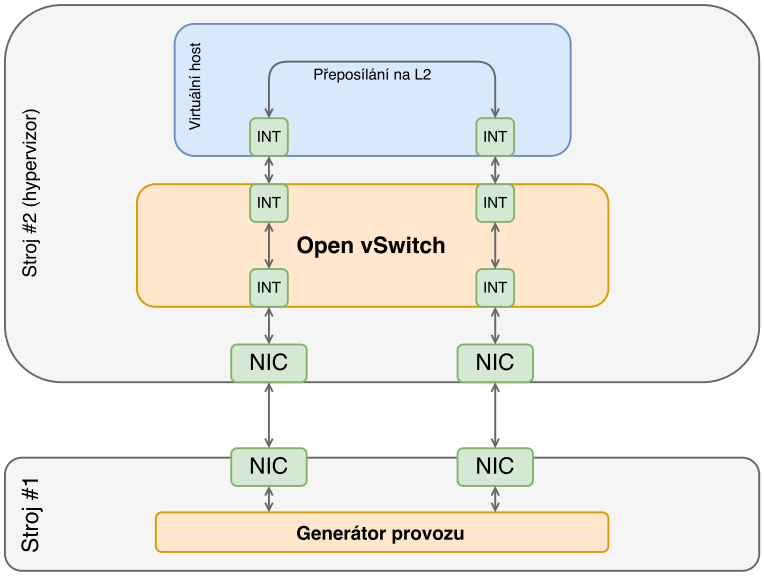
4.2.2 Topologie PVP

Na rozdíl od topologie P2P probíhá u PVP přeposílání rámců až na *úrovni virtuálního hosta*. Přepínač Open vSwitch zde zajišťuje přeposílání rámců vždy pouze mezi *fyzickým rozhraním síťové karty a virtuálním rozhraním hosta*. V tomto případě se Open vSwitch nechová jako klasický přepínač (nepracuje v tzv. learning módu). K přeposílání dochází na základě OpenFlow pravidel (viz příloha D). Topologie PVP je znázorněná na obrázku 4.2.

[3] Nejmenší rámce (rámce nerealisticky malé, obsahující minimální, popř. žádné místo na data) lépe charakterizují chování DUT. Režie, a tedy i zátěž DUT pro zpracování malých rámců, je tak o *mnoho* větší.



Obrázek 4.1: Schéma topologie P2P.



Obrázek 4.2: Schéma topologie PVP.

Kapitola 5

Automatizace testování výkonnosti Open vSwitch s DPDK

Jako u většiny software i zde vzniká potřeba virtuální přepínač Open vSwitch a knihovnu DPDK pravidelně testovat. Každá nová vývojová verze SW s sebou může přinést mnoho nových problémů. Proto je nutné testovat jak celkovou *funkčnost*, tak i jejich *výkon*.

Automatizací testování výkonu se zabývají následující odstavce práce. Systémy, které jsou zde zmiňovány, jsou prerekvizitou pro správnou funkčnost celého testování. Důležité je zmínit, že všechny tyto systémy jsou ve firmě Red Hat nasazeny v produkčním prostředí.

Proces průběžné integrace testování (CI)

Průběžná integrace (z angl. Continuous Integration - CI) je pojem pocházející ze softwarového inženýrství. CI je souhrn metod k urychlení vývoje software. Slouží především k rychlému nalezení nedostatků a chyb ve vývojové fázi SW. Základem procesu je průběžné (pravidelné) a časté vydávání nových verzí SW (tzv. integrace). Integrace je ověřena automatickými testy k co nejrychlejšímu nalezení chyb.

Základní součásti procesu CI:

- **Centrální úložiště zdrojových souborů** – Každý SW projekt se skládá z určitého množství zdrojových kódů. Tyto kódy jsou většinou uloženy v tzv. repozitáři spolu s kompletní historií změn.
- **Automatizace překladu zdrojových souborů** – Celý projekt obsažený v repozitáři by měl být jednoduše sestavitelný (nejlépe pomocí jediného příkazu). Výsledkem takového sestavení je tzv. *build*. Buildem může být například distribuční balíček (RPM, DEB, apod.).
- **Automatizace testování** – To, že sestavení projektu proběhlo úspěšně, ještě nemusí znamenat, že samotný SW neobsahuje žádné chyby. Je proto důležité zkontrolovat, že SW pracuje správně. K otestování funkcionality slouží testování jednotek (tzv. *unit testing*). Do této kategorie se ale kromě funkčních testů řadí i testy výkonnostní. Takové testy dávají smysl u aplikací, ve kterých je důležité zachování jejich výkonu napříč několika verzemi. Celý SW totiž může fungovat správně, jeho výkon se ale může v průběhu několika revizí zásadně změnit. Testy by se měly spouštět s *každým nově vytvořeným buildem*.

- **Kontrola kvality a zpětná vazba** – Tester hlídá, jestli testování probíhá v pořádku. Zároveň kontroluje podezřelé výsledky. V případě, že není s výsledky testů spokojen, nahlásí zjištěné problémy zpět vývojáři. Ten se problém pokusí nalézt a opravit. Po opravě se celý CI proces opakuje.

5.1 Systémy použité pro implementaci CI

Celý následující CI proces je inspirovaný již funkčními a plně nasazenými testy pro testování výkonnosti síťové vrstvy jádra. Implementačně je však *zcela* odlišný.

Proces CI testování postupně prochází několika systémy. Každý ze systémů disponuje svým XML-RPC API, pomocí kterého je možné systémy navzájem propojit. Všechny zmiňované systémy jsou kompletně open-source. V následující části jsou tyto systémy detailněji rozebrány. V sekci 5.2 je popsán celý CI proces detailněji.

5.1.1 Koji

Systém Koji, v rámci společnosti Red Hat také zvaný Brew, slouží k automatizovanému překladu a distribuci programových balíků. Systém se zaměřuje pouze na balíky typu RPM[1].

Koji interně využívá nástroj Mock, který zajišťuje překlad každého SRPM[2] balíku ve vlastním adresářovém stromu (s využitím chroot[3]). Díky tomu je možné vytvářet balíky programů pro různé architektury (pomocí cross-kompilace). Překlad a vytvoření balíků není vůbec závislý na architektuře a operačním systému hosta.

5.1.2 Jenkins

Jenkins je multiplatformní, velmi univerzální systém pro implementaci testování CI. Dokáže překládat SW, testovat ho, popř. umí automaticky informovat vývojáře o vniklých problémech. Spuštění nové úlohy v systému Jenkins může být vyvoláno různými událostmi - např. změnou kódu v repozitáři.

5.1.3 Beaker

V případě, že je potřeba sdílet fyzické (v určitých případech i virtuální) servery mezi více lidmi a používat je k různým účelům, je vhodné mít nástroj pro jejich centralizovanou správu a rezervaci. K tomu slouží právě systém Beaker, který je schopný na základě specifikovaných požadavků[4] automaticky vyhledat množinu systémů. Z ní je schopný jeden (nebo více) systémů zarezervovat a následně na nich spustit určité akce, zvané úlohy (z angl. tasks) [31].

Vyhledání a spuštění úloh na jednom či více systémů probíhá na základě tzv. *jobu*. Job se specifikuje v jazyce XML pomocí přesně definovaných konstrukcí. Každý z jobů se skládá z tzv. *receptů*. Každý recept popisuje jaké úlohy se budou na daném systému provádět. Job lze do systému odeslat buď manuálně pomocí formuláře nebo použitím utility `beaker-client`.

[1] RPM - Red Hat Package Manager - Balíčkovací systém, původně vyvinutý pro linuxovou distribuci Red Hat Linux, který je dnes používán v mnoha distribucích jako CentOS, Fedora nebo Scientific Linux.

[2] SRPM - Zdrojový soubor, ze kterého je vytvořen konečný RPM balík.

[3] Systémové volání, které slouží ke změně kořenového adresáře pro určitý proces a jeho případné potomky.

[4] Požadavky mohou být různého typu - na architekturu systému, velikost diskového prostoru, velikost RAM, popř. požadavky na určitý typ periferního zařízení.

Před spuštěním jakékoliv úlohy musí být všechny její soubory nejprve nainstalovány na cílový systém. Instalace probíhá prostřednictvím předgenerovaného RPM balíčku. Po instalaci musí pouze Beaker úlohu spustit. Úloha není nic jiného než množina spustitelných souborů. Její spuštění probíhá pomocí programu `make` na základě souboru `Makefile`. Systém Beaker v adresáři úlohy zavolá nejprve pravidlo `make build`, v rámci kterého jsou nastavena práva pro spuštění pro určité soubory. V druhé fázi zavolá Beaker pravidlo `make run`. To už většinou spustí pouze soubor `testrun.sh`, který je vstupním bodem celé úlohy.

5.1.4 Nástroj VSperf

Projekt VSperf (také nazýván vSwitchPerf) vznikl v rámci iniciativy OPNFV. Jedná se o otevřený, vysoce konfigurovatelný, modulární nástroj, kompletně napsaný v jazyce Python [37].

VSperf si bere za cíl přesně *definovat* a *implementovat* referenční platformu pro testování NFV, která bude zajišťovat nejen konzistenci a funkčnost jednotlivých komponent, ale i výkonnost. Zjednodušeně se jedná o framework, který zajišťuje konfiguraci virtuálního přepínače na systému DUT, spuštění testů (včetně generátoru provozu) a zaznamenání výsledků a stavu systému [29]. V případě provádění testů, u kterých dochází k přeposílání na úrovni virtuálního hosta, si VSperf jeho spuštění a konfiguraci řeší ve vlastní režii.

Důvody pro vznik projektu VSperf jsou především následující:

- Prováděná nastavení virtuálního přepínače, parametrů DPDK knihovny nebo nastavení systému DUT mohou být *vysoce* variabilní. Tato variabilita může podstatným způsobem ovlivnit změřený výkon, popř. stabilitu získaného výsledku.
- V případě, že by si každý napsal vlastní testy, které by komponenty a systém na DUT konfigurovaly vlastním způsobem, bylo by velmi těžké získat konzistentní výsledky.
- Díky vysoké variabilitě nastavení prostředí (viz první bod) je dobré mít možnost vše konfigurovat přehledně na jednom místě použitím konfiguračního souboru. Tento přístup umožňuje jednoduše a *rychle* změnit nastavení prostředí, ve kterém testování probíhá.
- Je dobré, aby potenciální zákazníci měli možnost výsledky měření *reprodukovat*. Není dobré, když firma prodává hotový produkt, ke kterému dodává výsledky výkonnostního měření a zákazník tyto hodnoty nemá možnost sám ověřit.

Všechny zmíněné problémy VSperf efektivně řeší. Díky němu není problém spustit testy jak na různém hardware, tak například na jiném virtuálním přepínači než Open vSwitch. Také není problém spustit více různých typů testů pro jednu určitou konfiguraci systému a jeho komponent. Stačí k tomu pouze minimální změny v konfiguračních souborech. Není nijak nutné složitě upravovat testovací skripty.

Pozn.: Do oficiální verze projektu bylo v rámci vývoje automatizovaného prostředí několikrát přispěno. Bylo opraveno několik zásadních chyb, které způsobovaly problémy při jeho spuštění. Zároveň byl upraven formát výstupních dat tak, aby byl lépe strojově zpracovatelný.

5.1.5 MoonGen

Jako generátor provozu pro implementaci automatizovaného testování výkonu a i pro následné měření výkonu byl zvolen MoonGen. Jedná se o otevřený, vysokorychlostní a plně programovatelný generátor provozu. Dle dokumentace je generátor s využitím technologie DPDK schopný saturovat 10Gbit linku ethernetovými rámci o minimální velikosti (64 B bez přítomnosti značení 802.1Q) s využitím pouze jednoho CPU jádra[5] [9].

Celý generátor je postavený na knihovně `libmoon`, která zajišťuje vysokoúrovňové rozhraní nad knihovnou DPDK - především zjednodušuje její inicializaci, inicializaci paměti a periferních zařízení. Své rozhraní vystavuje pro jazyky `Lua` a `C/C++`. Díky tomu je možné kontrolovat výstupní data generátoru pomocí uživatelem definovaného Lua skriptu.

Uživatelský skript se dělí na dvě části - *master* a *slave*. Každá z těchto částí v rámci systému běží ve vlastním vlákně (z angl. thread). Master vlákno zajišťuje načtení dodatečné konfigurace, konfiguraci síťové karty a jejích front. Následně pro každou z front vytvoří nové slave vlákno, které spustí. Jednotlivá slave vlákna zajišťují generování dat.

Na první pohled by se mohlo zdát, že interpret dynamického jazyka Lua nebude dostatečně rychlý a bude celé generování dat zpomalovat. To generátor MoonGen řeší použitím vysokorychlostního interpretu jazyka Lua - LuaJIT (z angl. Lua Just-In-Time překladač) napsaného v jazyce C a assembleru. Jeho implementace zpracování jazyka Lua je prezentována jako jedna z nejrychlejších vůbec [21].

Nutno dodat, že jazyk Lua nedisponuje nativní podporou pro multi-threading. Tento nedostatek MoonGen řeší spuštěním virtuálního LuaJIT stroje *pro každé vlákno*. Tato vlákna spolu sdílejí pouze svůj stav, a to pomocí speciální podpůrné vrstvy implementované přímo v generátoru MoonGen.

MoonGen zároveň umí generovaná data i přijímat - umí například zkontrolovat, jestli se některé ethernetové rámce při cestě přes DUT ztratily nebo nepoškodily. Příjem dat je opět v plné kontrole uživatelem definovaného Lua skriptu.

Na obrázku 5.1 je pro přehlednost znázorněna architektura generátoru MoonGen.

Implementace RFC2544 v generátoru MoonGen

Generátor MoonGen ve svém základu neposkytuje skript pro měření výkonu pomocí metodologie popsané v RFC2544 (viz 4.1). Tento skript musel být vytvořen. O to se postarali vývojáři Andrew Theurer a Bill Michalovski. Ti implementovali skript `trafficgen.lua`, který pomocí *metody půlení intervalu* hledá takovou maximální rychlost odesílání rámců R tak, aby jejich procentuální ztrátovost Z byla *nulová*, popř. *menší než specifikovaná* (parametr `acceptableLossPct`).

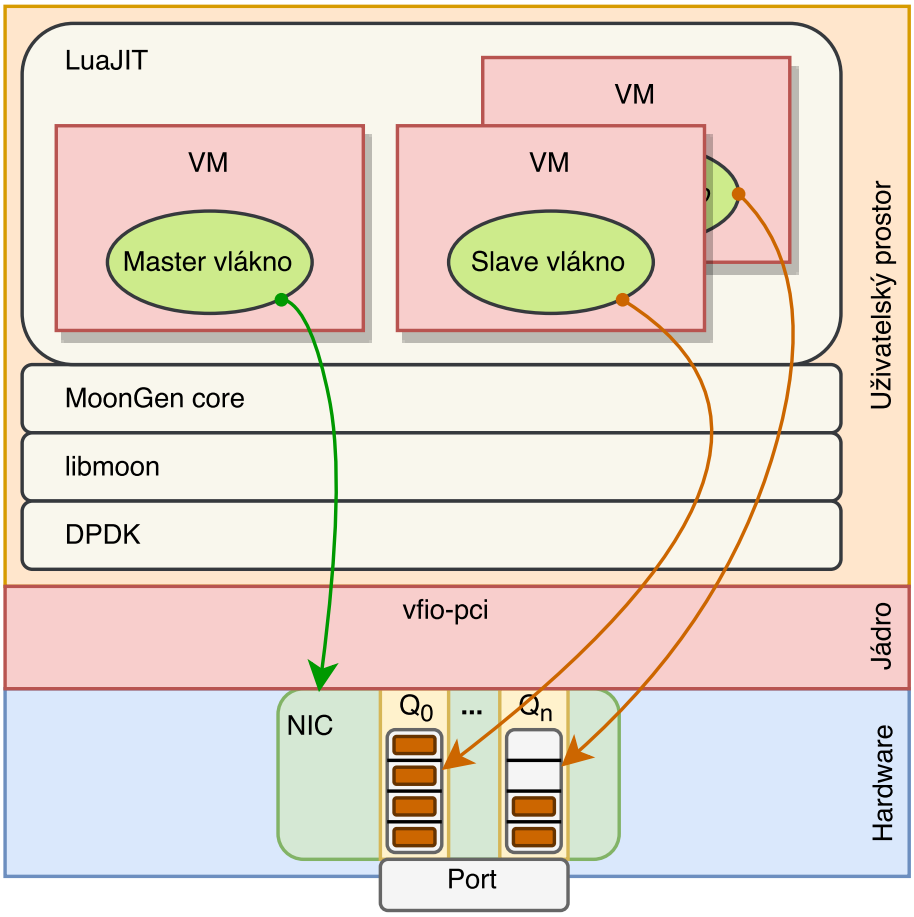
Jakmile rychlost R najde, spustí tzv. test validity. Při něm generátor odesílá rámce rychlostí R, ale už neprovádí žádné půlení intervalu. Pouze kontroluje, že ztrátovost Z nepřesáhla definované meze. Ztrátovost Z se spočte vzorcem 5.1, kde N_s je počet rámců odeslaných generátorem provozu a N_r je počet rámců přijatých generátorem.

$$Z = \frac{N_s - N_r}{N_s} \tag{5.1}$$

Skript je plně konfigurovatelný. Nejdůležitějšími parametry jsou:

- `startRate` – počáteční rychlost odesílání rámců. V případě 10Gbit linky testy začínají na rychlosti 14.88 Mpps.

[5] Na procesoru Intel Xeon E5-2620v3 při frekvenci 1.5 GHz se síťovými kartami Intel X540.



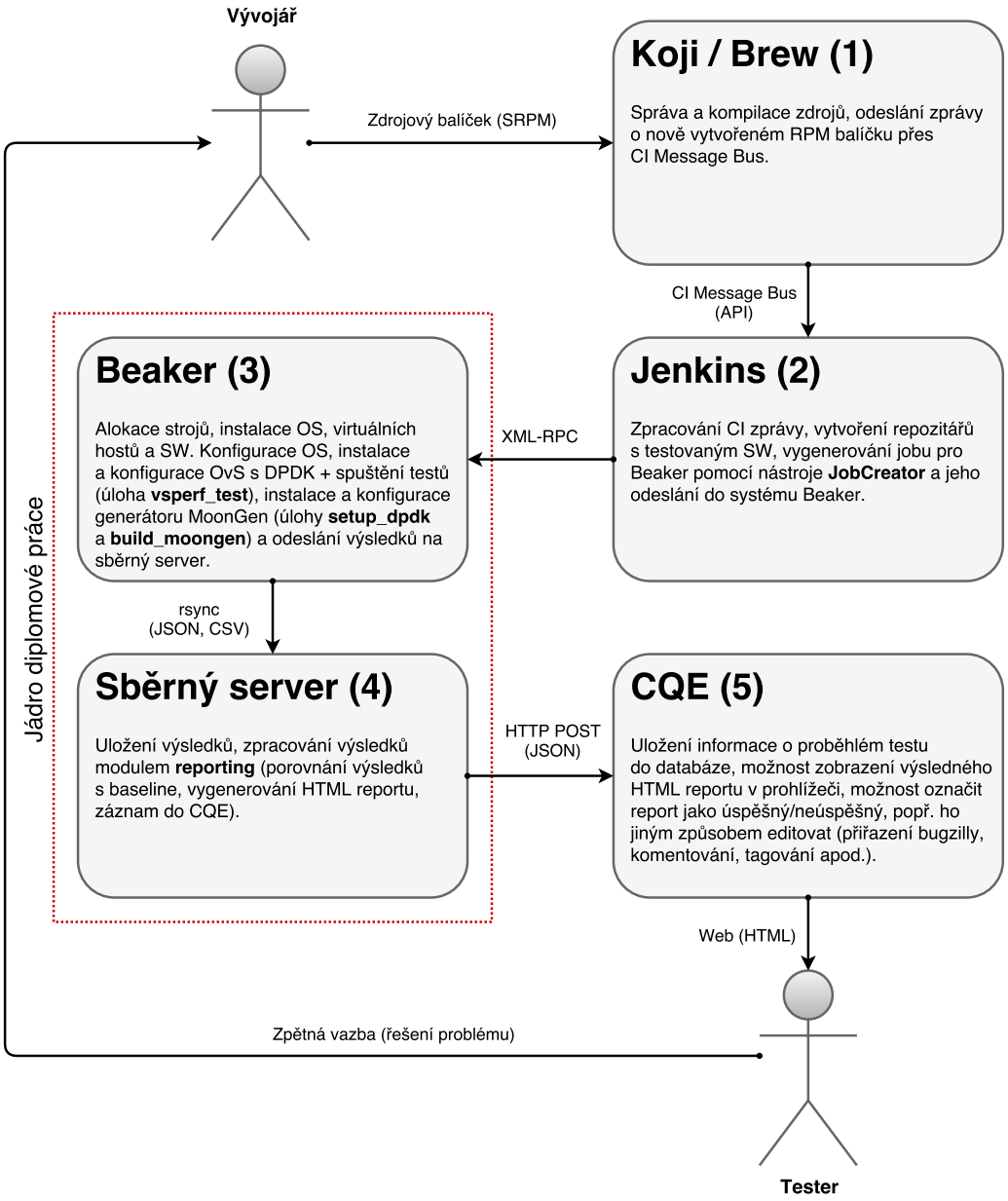
Obrázek 5.1: Architektura generátoru MoonGen.

- `runBidirec` – určuje, jestli se bude testovat v obou směrech.
- `searchRunTime` – počet sekund, po který je prováděn test při rozpůlení intervalu.
- `validationRunTime` – počet sekund, po který se po nalezení rychlosti R provádí konečný test validity. Tato doba je zpravidla několikanásobně delší než `searchRunTime`.
- `acceptableLossPct` – maximální povolená ztrátovost rámců, při které je test shledán úspěšným. Je udávaná v procentech.
- `frameSize` – velikost ethernetového rámce v bajtech (udávaná včetně CRC kontrolního součtu).
- `loggingLevel` – v rámci této práce byl skript rozšířen o možnost specifikace úrovně logování[6]. Při dlouhých testech docházelo k vytvoření nepřiměřeně velkých souborů, které obsahovaly nepotřebné ladicí informace. Díky této volbě lze tyto informace omezit pouze na ty potřebné.

[6] Viz https://github.com/atheurer/lua-trafficgen/pull/22

5.2 Průchod systémy v rámci CI procesu

Celý CI proces před samotným testováním prochází několika důležitými systémy. Na obrázku 5.2 lze vidět zjednodušený pohled na celý CI proces. Červený obdélník v obrázku znázorňuje nejvýznamnější části procesu, které byly v rámci této práce implementovány.



Obrázek 5.2: Zjednodušený pohled na CI proces.

Detailnější schéma celého CI procesu, znázorněného pomocí *UML diagramu aktivit*, lze nalézt v příloze A.

5.2.1 Získání SW pro účely testování

Jakmile vývojář, popř. správce SW balíku, odešle novou verzi zdrojového SW balíku (SRPM) do systému Koji, systém se pokusí balík sestavit a vytvořit z něho balík RPM.

Zde na řadu vstupuje systém Jenkins. Ten naslouchá na sdílené sběrnici pro zasílání zpráv tzv. *CI Message Bus*. Na tuto sběrnici systém Koji zasílá zprávy o každé prováděné akci. V případě testování Open vSwitch a DPDK nás budou zajímat pouze zprávy o úspěšně sestavených balících pouze od specifického uživatele a tagu. Dále je důležitý identifikátor úlohy, v rámci které byl balík sestaven (tzv. ID úlohy).

Filtrování zpráv Jenkins provádí na základě konfiguračního souboru (soubor je součástí diplomové práce). Pravidla pro jejich filtrování jsou specifikovaná v části *triggers*. Pokud zpráva splňuje všechny podmínky, vykoná Jenkins hlavní část konfiguračního souboru - část *builders*. Ta obsahuje Bash skript, který zajišťuje přípravu repozitáře obsahujícího testovaný SW (přesněji obsahuje přeložené RPM balíčky získané na základě ID úlohy, která SW přeložila) a vygenerování a odeslání XML jobů do systému Beaker pomocí nástroje JobCreator.

5.2.2 Vygenerování jobu pro systém Beaker

Vygenerování zdrojového jobu ve formátu XML zajišťuje samostatně spustitelný skript zvaný JobCreator. Tento skript byl vytvořen v rámci této práce. Při jeho vývoji bylo dbáno na jeho univerzálnost, rozšiřitelnost a přehlednost. Na těchto vlastnostech má velký podíl jeho objektová architektura.

Skript intuitivně zajišťuje generování jobů dle přesně zadaných kritérií. Generování probíhá na základě předem definovaných šablon nacházejících se v modulu `recipesets.py`. V tomto modulu má každý testovaný scénář definovanou svou vlastní šablonu. Šablony jsou definovány přímo v jazyce Python. Díky tomu lze chování šablon přesně programově ovlivňovat. Šablony jsou interně dále dekorované pomocí objektového návrhového vzoru *dekorátor*. Dekorátory jsou definované v modulu `decorators.py`.

JobCreator na svém vstupu přijímá několik povinných parametrů:

- `--host` – specifikuje jednoho nebo více hostů pomocí jejich *hostname*.
- `--distro` – určuje distribuci, na které se bude testování provádět.
- `--config` – specifikuje testovací scénář. Na jeho základě JobCreator vygeneruje job s úlohama v přesně specifikovaném pořadí a s přesně definovanými parametry. Dále tato volba určuje, co se bude testovat. Popis podporovaných testovacích scénářů lze nalézt v sekci 5.3.1.

JobCreator může přijímat i parametry volitelné. Jejich popis lze nalézt zavoláním příkazu `jobcreator --help`.

V ukázce 5.1 lze vidět příklad vygenerování XML jobu, který zajistí instalaci distribuce *Red Hat Enterprise Linux 7* na dvou serverech. V tomto případě bude první server sloužit jako generátor provozu, druhý bude stroj testovaný (DUT). Konfigurace `OVS_DPDK1q_GUEST_DPDK1q` zajistí vytvoření specifického prostředí pro testování.

```
./jobcreator \
    --host      "moongen.company.example.com" \
    --host      "dut.company.example.com" \
    --config    "OVS_DPDK1q_GUEST_DPDK1q" \
    --distro    "RHEL-7.3" \
    --kickrepo "http://company.example.com/repos/dpdk/16.11/4.el7" \
    --kickrepo "http://company.example.com/repos/openvswitch/2.6.1/15.el7" \
    --tag       "Toto je oznaceni generovaneho jobu" \
    --reserve 100 \
    --autosubmit
```

Kód 5.1: Ukázka spuštění skriptu pro vygenerování XML jobu pro systém Beaker.

Jak vypadá hotový vygenerovaný job lze vidět v ukázce v příloze B.

5.3 Testování v systému Beaker

Jakmile systém Beaker obdrží nově vygenerovaný job, zařadí jej do fronty. V této frontě bude job čekat tak dlouho, dokud se neuvolní zdroje, které požaduje.

Jakmile jsou zdroje dostupné, job je spuštěn a začnou se provádět jeho recepty. Každý z receptů představuje jeden systém a jeho úlohy, které popisují akce spouštěné na těchto systémech. Spuštění úloh v jednotlivých receptech v rámci jednoho jobu probíhá paralelně. Proto je nutná jejich synchronizace. V následujících odstavcích jsou tyto úlohy rozebrány detailněji.

5.3.1 Instalace a spuštění testů

Mezi počáteční úlohy spouštěné na každém ze strojů patří instalace operačního systému. Tu provádí úloha `/distribution/install`. V rámci této úlohy jsou na systém nainstalovány potřebné SW balíky a jsou nakonfigurovány *parametry jádra*. Jádro je nutné startovat s podporou pro IOMMU (viz 3.1.1) a se správně nakonfigurovanými velkými stránkami (viz 3.1.1). Nakonec je celý systém ještě restartován.

Další úlohou (spouštěnou pouze v určitých případech) je vytvoření, instalace a konfigurace virtuálního hosta. Tyto akce Beaker provádí na základě tzv. *guest receptu* (viz dále).

Ihned po tom, co jsou systémy nainstalovány, jsou spuštěny úlohy specifické. Mezi ně patří konfigurace dodatečných repozitářů, instalace potřebného SW a především spuštění výkonnostních testů.

Synchronizace

Jelikož se testování účastní více než jeden stroj, potřebujeme synchronizovat spouštění úloh na jednotlivých strojích. Systém Beaker ve svém základu poskytuje rozhraní pro synchronizaci úloh [31]. Toto rozhraní ale v určitých případech není dostačující, popř. pro účely testování Open vSwitch s DPDK nefunguje správně[7]. Proto testy využívají alternativní implementaci synchronizace vytvořenou Ing. Adamem Okuliarem. Tato synchronizace je spouštěna v rámci úlohy `/performance/perf_synchronization`.

[7] Při konfiguraci sítě na DUT docházelo k zamrznutí celé synchronizace. Systémy zůstávaly rezervované a žádná z úloh se na nich neprováděla.

Příprava generátoru provozu MoonGen

V rámci prvního receptu probíhá instalace generátoru provozu MoonGen a příprava systému pro jeho spuštění. Tu zajišťují následující úlohy:

- `/performance/setup_dpdk` – Zajistí izolaci jader CPU, která budou vyhrazena pro polling síťových karet (poběží na nich PMD vlákna 3.1.2). Jádra jsou izolována jak od procesů, tak i od přerušení. Izolaci jader od plánování plánovačem OS zajišťuje program zvaný `tuned`. Ten izolaci provádí na základě konfiguračního souboru `cpu-partitioning-variables.conf`. K izolaci pak interně dochází využitím parametru jádra `isolcpus` (více viz 3.1.1). Nutné je izolovat i přerušení. O ta se stará nástroj `irqbalance`.

  Dále tato úloha zajišťuje zavedení potřebných modulů do jádra - především modulu `vfio-pci`[8]. Jakmile je tento modul zaveden, může úloha provést binding karet pod DPDK (více viz sekce 3.1.2).

  Alokaci velkých stránek potřebných pro správnou funkčnost DPDK si MoonGen řeší interně sám.

- `/performance/build_moongen` – Získá MoonGen z jeho repozitáře, přeloží ho a nainstaluje do systému. Zároveň na systému vytvoří konfiguraci DPDK, kterou generátor načítá. V té jsou specifikována CPU jádra, na kterých poběží PMD vlákna knihovny DPDK. Zároveň jsou zde pomocí PCI identifikátoru specifikované síťové karty, které mají být využívané k odesílání generovaných dat.

Obě úlohy lze spustit zcela samostatně (nezávisle na systému Beaker). Spuštění se provádí zavoláním skriptu `runtest.sh`, který se nachází v kořenovém adresáři obou těchto úloh.

Příprava testovaného systému

Na straně DUT se musí nakonfigurovat daleko více než na straně generátoru. Jednou ze stěžejních částí je instalace a konfigurace virtuálního hosta. Ta se provádí opět za pomoci systému Beaker, který je schopný virtuálního hosta automatizovaně nainstalovat a spustit na něm potřebné úlohy. To se provádí na základě předpisu zvaného *guestrecipe*, který je přímou součástí jobu. Na virtuálního hosta je v rámci guest-receptu nainstalován operační systém, knihovna DPDK a další potřebné balíčky. O konfiguraci (celého DUT i virtuálního hosta), se stará úloha `/performance/vsperf_test` v rámci svého parametru `setup --guest`.

Jak v případě DUT, tak i virtuálního hosta, tato úloha instaluje a perzistentně[9] konfiguruje `tuned` (izolace jader CPU), izoluje přerušení (`irqbalance`) a vypíná NetworkManager, který by mohl v systému provádět nežádoucí nastavení sítě. Dále na DUT nainstaluje privátní SSH klíč pro vzdálené ovládání systému s nainstalovaným generátorem provozu MoonGen. Po konfiguraci je celé DUT restartováno. Po restartu se na DUT znovu spustí úloha `/performance/vsperf_test`, tentokrát ale za účelem nastavení neperzistentních konfigurací a spuštění testů. Virtuální host zůstává vypnutý a dále se pomocí systému Beaker nijak nezapíná ani nekonfiguruje. Jeho spuštění a nastavení neperzistentních konfigurací si řeší testy samy v rámci spuštění VSperfu (viz 5.1.4).

[8] Tento modul poskytuje rozhraní pro práci s IOMMU z uživatelského prostoru.
[9] Provádí nastavení, která jsou zachována při restartu systému.

Stejně jako v případě úlohy pro instalaci generátoru provozu lze i úloha `vsperf_test` spustit kompletně samostatně. Spuštění zajišťuje skript `runtest.sh`. Parametry nutné pro spuštění skript přijímá ze systémové proměnné `RUN_VSPERF_ARGS`. Příklad manuálního spuštění úlohy lze vidět v ukázce 5.2.

```
RUN_VSPERF_ARGS="--log-level debug setup --guest OVS_DPDK1q_GUEST_DPDK1q"
./runtest.sh
```

Kód 5.2: Manuální spuštění úlohy `vsperf_test` ve virtuálním hostovi.

Spuštění testů

Pokud se všechny předešlé úlohy podařilo provést bez problému, mohou být spuštěny samotné testy. Spuštění testů probíhá v rámci úlohy `/performance/vsperf_test`. Zde je nutné přesně rozlišit testovací scénář. Název testovacího scénáře je úloze předáván pomocí parametru přímo ze systému Beaker (jedná se o stejné jméno testovacího scénáře, které bylo na počátku předáváno JobCreatoru (viz 5.2.2) při generování jobu).

Podporované scénáře a jejich parametry lze vidět v následující tabulce. Sloupec **fronty** představuje počet front síťové karty použitých k testování (tzv. *multiqueue*[10]).

Jméno scénáře	**Přeposílání**	**Fronty**	**DPDK**
`OVS_DPDK1q_GUEST_DPDK1q`	P2P (Open vSwitch), PVP (testpmd)	1TX, 1RX	Ano - na obou úrovních.
`OVS_VANILLA_GUEST_linuxbridge`	P2P (Open vSwitch), PVP (Linux-Bridge)	1TX, 1RX	Kompletně bez DPDK.
`OVS_DPDK1q_GUEST_linuxbridge`	P2P (Open vSwitch), PVP (Linux-Bridge)	1TX, 1RX	Pouze na úrovni OvS.
`OVS_DPDK2q`	P2P (Open vSwitch)	2TX, 2RX	Pouze na úrovni OvS.
`OVS_DPDK4q`	P2P (Open vSwitch)	4TX, 4RX	Pouze na úrovni OvS.
`OVS_DPDK2q_GUEST_DPDK2q`	P2P (Open vSwitch), PVP (testpmd)	2TX, 2RX	Ano - na obou úrovních.

Na základě jména testovacího scénáře a hostname DUT provede úloha vsperf_test vygenerování konfigurace pro nástroj VSperf (viz 5.1.4). Konfigurace je generovaná z několika hierarchicky uspořádaných konfiguračních souborů přesně na míru daného systému. Příklad vygenerovaného konfiguračního souboru pro scénář `OVS_DPDK1q_GUEST_DPDK1q` lze najít v přílohách C.

[10] Některé novější karty poskytují pro přenos dat více odesílacích (TX) a přijímacích (RX) front. Síťová karta má pak možnost přijímaná data vkládat do více RX front (většinou použitím hašovací funkce). Každá RX fronta vyvolává jiné přerušení. Tato technika se nazývá multiqueue. Díky tomu je možné tato přerušení distribuovat mezi různá procesorová jádra a přicházející data zpracovávat *paralelně* (tzv. receive-side scaling). Stejně tak lze pro zvýšení výkonu odesílání dat využít více TX front přiřazených k více různým procesorovým jádrům (tzv. transmit packet steering).

V další fázi běhu zajišťuje úloha vsperf_test spuštění testovacího frameworku VSperf. VSperf je spouštěn s parametrem `--config`, který specifikuje umístění dříve vygenerovaného konfiguračního souboru. Dále už se o vše stará přímo VSperf.

VSperf na začátku ověří, že jsou do jádra zavedeny všechny potřebné moduly. Pokud nejsou, zavede je, provede binding karet specifikovaných v konfiguračním souboru pod DPDK a připojí souborový systém pro práci s velkými stránkami *hugetlbfs*. Jako další spustí na pozadí služby potřebné pro správnou funkčnost virtuálního přepínače Open vSwitch (`ovs-vswitchd` a `ovsdb-server`), vytvoří novou databázi pro uchování jeho konfigurace a nakonfiguruje na něm testovanou síťovou topologii pomocí OpenFlow. Příklad konfigurace pravidel lze vidět v příloze D.

V případě, že se testuje topologie PVP, nastartuje VSperf pomocí emulátoru Qemu virtuálního hosta, který byl vytvořen systémem Beaker. Boot a dodatečná nastavení na běžícím hostovi (vypnutí firewallu, zavedení modulů do jádra, binding karet pod DPDK aj.) si VSperf řeší kompletně sám.

Jakmile jsou DUT a i jeho případný virtuální host připraveny, přihlásí se VSperf pomocí SSH na systém s nainstalovaným generátorem provozu MoonGen. Na vzdáleném systému pomocí konfiguračního souboru nastaví parametry RFC2544 testu (viz 5.1.5) a celý generátor spustí.

Poté co MoonGen dokončí měření, vypíše změřená data na svůj standardní výstup. Výstup VSperf zaznamená a na jeho základě vygeneruje celkový výsledek testu. Výsledkem testu je adresář, kteřý obsahuje soubory popisující:

- Informace o systému DUT - operační systém, informace o zakladní desce, typ CPU, počet jeho jader, počet jader využitých k testování, typ a frekvence operační paměti.
- Síťové karty použité k testování (fyzické i virtuální), jejich výrobce, verze firmware, PCI slot do kterého jsou připojené apod.
- Verze testovaného SW (jak pro DUT, tak pro případného virtuálního hosta) - verzi jádra, přepínače Open vSwitch, verzi knihovny DPDK, verzi nástroje VSperf a verzi generátoru MoonGen.
- Boot parametry DUT i případného virtuálního hosta.
- Typ použitého generátoru provozu a jeho verzi.
- Typ provedeného testu, typ provozu (vždy obousměrný) - UDP, TCP, apod.
- Propustnost v jednotkách FPS (frame per second).
- Velikost testovaných rámců - fixní velikosti, popř. IMIX[11].
- Doba běhu testu.
- Data z průběhu testování - čas strávený na CPU, obsazenost paměti (především pro aplikace ovs-vswitchd a ovsdb-server).

[11]IMIX (Internet Mix) - Představuje typický internetový provoz procházející přes síťová zařízení. Tento typ provozu je využíván k simulaci prostředí reálného provozu na Internetu.

Ihned po zaznamenání všech zaznamenaných dat VSperf postupně ukončí služby, které na počátku spustil, zastaví případného virtuálního hosta a řádně se ukončí. Všechna zmíněná data jsou následně odeslána pomocí `rsync` na sběrný server, kde dochází k jejich dalšímu zpracování.

Alokované stroje jsou v systému Beaker uvolněny a mohou být použity i pro jiné účely než pouze testování.

5.4 Automatizované vyhodnocení výsledků

Úloha `/performance/vsperf_test` zajistí pouze získání surových dat z průběhu testování. Tato data musí být zpracována, porovnána a zobrazena vhodným přehledným způsobem. Právě o to se stará modul `reporting`, který byl v rámci této práce implementován.

Modul `reporting` se skládá ze dvou skriptů:

- `generate-report` – přijímá na vstupu syrová data jednoho nebo více výsledků. Skript data porovná a vygeneruje celkový report.
- `process-ci-results` – slouží pouze ke kompletní automatizaci generování reportů pro nově příchozí výsledky. Ten v určitých časových intervalech (pomocí nástroje Cron) kontroluje specifický adresář (`ci_in`). V případě, že se v něm objeví nový výsledek, zavolá `generate-report`, pomocí kterého *porovná daný výsledek s výsledkem referenčním* (tzv. baseline). Baseline se nevolí automaticky, ale manuálně nastavením proměnné přímo ve skriptu. Po porovnání je zpracovaný výsledek přesunut do adresáře se zpracovanými výsledky - `ci_processed`.

Jako baseline je vždy zvolen výsledek testu předchozí stabilní verze knihovny DPDK a přepínače Open vSwitch.

5.4.1 Generování výsledného reportu

Generátor reportu je kompletně implementovaný v jazyce Python. Při svém spuštění očekává povinný parametr `--baseline`, který specifikuje výsledek použitý jako referenční pro porovnávání s jinými. Dále očekává parametr `--output-dir`, který slouží ke specifikaci adresáře, do kterého bude vygenerován konečný report. Skriptu lze zadat i nepovinný seznam výsledků (jako tzv. poziční argumenty), které budou porovnány s výsledkem referenčním. Pro bezchybné porovnání skriptem musí být výsledky zadávané na vstupu specifikovány pomocí cesty k jejich adresáři. Zde je nutné zdůraznit, že jméno adresáře se surovými daty z testování nesmí být manuálně měněno. Skript počítá s tím, že jméno adresáře bude přesně takové, jaké výsledku přiřadila úloha `/performance/vsperf_test` na konci měření.

Skripty pro porovnávání se skládají ze dvou modulů - `loaders.py` a `reports.py`. Modul `loaders.py` zajišťuje načtení surových výsledků do paměti. Interně je každý z výsledků rozdělen na jednotlivé testy programu VSperf (jeden výsledek v sobě může obsahovat testy i pro několik testovaných topologií). Každý z těchto testů se pak ještě dělí na jednotlivé běhy generátoru provozu (např. jeden běh pro každou testovanou velikost rámce).

Modulu `loaders.py` následně využívá modul `reports.py`, jehož úkolem je načtená data porovnat a zobrazit pomocí tzv. reportu. Report není nic jiného než adresář obsahující základní logy, parametry prostředí, porovnaná data ve formě grafů a HTML soubor `index.html` zobrazující grafy a hodnoty změřené propustnosti přehledně v tabulkách.

Reporty jsou generovány na základě předem specifikovaných Jinja2[12] šablon umístěných v adresáři `templates`. Pro generování grafů byla použita knihovna `matplotlib`[13].

Příklad reportu, ve kterém je porovnán výsledek se svou baseline, lze vidět v příloze E.

Ihned po vygenerování a uložení všech souborů reportu je nutné informaci o nově vzniklém porovnání zaznamenat. Zároveň je vhodné mít možnost dané porovnání okomentovat, popř. mu přiřadit číslo bugzilly[14]. K tomu slouží systém zvaný CQE (viz dále).

5.4.2 Uložení informace o výsledném reportu v systému CQE

Systém CQE (název vycházející z ClusterQE) je relativně nový interní systém společnosti Red Hat pro záznam výsledků testů. Celý je napsán v jazyce Python jako aplikace webového aplikačního frameworku Django.

CQE představuje finální místo celého CI procesu. V něm se shromažďují všechny výsledky měření. Vyhodnocení správnosti výsledku je ověřováno manuálně testerem - není tedy plně automatické. Je to především z toho důvodu, že občas dochází k chybám měření, které jsou většinou způsobeny krátkodobým výpadkem infrastruktury. Vyhodnocení výsledku je testerovi ulehčeno barevným zobrazením procentuální odchylky od referenčních hodnot (viz tabulka v příloze E obrázek E.1).

Jakmile tester vyhodnotí výsledek, může ho v CQE po přihlášení označit jako úspěšný nebo neúspěšný. Pokud tester není s výsledkem spokojený, může výsledek ověřit tím, že test spustí znovu, popř. přímo nahlásí problém (tzv. bug) v systému Bugzilla. ID nově přidaného problému lze v CQE chybnému reportu přiřadit.

Pozn.: Projekt CQE není dílem této práce. Do projektu bylo pouze zasláno několik vylepšení - úprava celkového vzhledu (CSS, HTML, generované formuláře), úprava velké části kódu dle doporučení PEP 8[15] a vytvoření skriptu `parse_wiki.py` pro převod výsledků ze staré databáze do CQE.

[12] Jinja2 - Šablonovací systém pro jazyk Python vycházející ze syntaxe Django - http://jinja.pocoo.org.
[13] http://matplotlib.org
[14] Bugzilla - Systém pro záznam a sledování problémů vzniklých v určitém SW.
[15] https://www.python.org/dev/peps/pep-0008

Kapitola 6

Měření výkonnosti různých konfigurací Open vSwitch

Postupem času jsou na virtuální přepínače (obecně na virtuální síťové funkce) kladeny stále větší a větší nároky. Dalo by se říci, že výkonnostně by měly stačit hardwarovým přepínačům, směrovačům apod. Měly by zvládat pracovat ve vysokých rychlostech - 25 GbE, 40 GbE nebo dokonce 100 GbE. Zároveň by na rozdíl od HW zařízení měly být jednoduše *škálovatelné* a *rozšiřitelné* o další funkcionalitu. Bohužel virtuální přepínače nikdy nebyly navržené na vysokorychlostní zpracování především malých dat, které typicky využívají velké telekomunikační společnosti (tzv. telco operátoři). Tito operátoři mají velké nároky na *nízkou latenci* a *vysokou propustnost* [29].

Následující sekce popisují prostředí, ve kterém byly prováděny výkonnostní testy pomocí automatizačních nástrojů popsaných v kapitole 5. Popisují výsledky měření propustnosti navržených topologií s využitím knihovny DPDK i bez ní.

6.1 Hardware použitý k měření

Na testování byly vždy použity dva stroje. Oba byly zcela identické. V každém z nich se nacházela jedna 10Gbit 2-portová síťová karta. Tyto karty byly vzájemně propojené metalickými SFP kabely. Optika zde nebyla použita především z důvodu vyšších pořizovacích nákladů. Zároveň by ani nebyl využit její plný potenciál, který spočívá v možnosti komunikovat na velkou vzdálenost. Stroje spolu totiž při testování sousedily v rámci jednoho racku.

Parametry strojů lze nalézt v následující tabulce:

`Typ systému:`	Lenovo x3550 M5
`Operační paměť:`	64 GiB (32 GiB na NUMA uzel)
`Disk:`	128 GiB (SSD)
`Počet soketů:`	2
`Procesor:`	Intel Xeon E5-2620v3 (2.40 GHz)
`Architektura:`	x86_64
`Počet CPU jader:`	6 jader na procesor (12 ze zapnutým HT)
`Intel Hyper-Threading Technology:`	zapnuto
`Intel Turbo Boost Technology:`	zapnuto
`Podpora pro virtualizaci:`	zapnuta (technologie VT-d i VT-x)
`Síťová karta:`	Dual-port 10 GbE Intel 82599ES
`Kabely použité pro propojení:`	Metalické Juniper EX-SFP-10GE-DAC-3M

6.2 Konfigurace testovaného systému (DUT)

Všechna měření byla prováděna na operačním systému Red Hat Enterprise Linux 7.3 (Maipo) s linuxovým jádrem verze `3.10.0-514.el7.x86_64`. V následujících sekcích jsou popsána nastavení specifická. Jsou zde popsány rozdíly nastavení při testování bez knihovny DPDK i s ní.

6.2.1 Nastavení systému při testování bez knihovny DPDK

Při testování přepínače Open vSwitch **bez DPDK** nebyly na systému DUT nastavovány žádné speciální volby. K testování byly použity nativní ovladače linuxového jádra. Pro komunikaci s fyzickými kartami 82599ES byl použit ovladač `ixgbe` (verze `4.4.0-k-rh7.3`). Verze firmware karty byla `0x61c10001`.

Žádná CPU jádra nebyla v průběhu testování nijak izolována. Technologie multiqueue (viz 5.3.1) zapnuta nebyla (pro komunikaci se síťovou kartou se vždy využívala jedna TX a jedna RX fronta). Chování přerušení také nebylo nijak programově měněno. Správa distribuce přerušení byla vždy přenechána na nástroji `irqbalance`.

Nastavení tzv. offloadů[1] nebylo měněno. Hodnoty nejvýznamnějších offloadů nastavených na fyzických rozhraních lze vidět v následující tabulce:

[1] Offloading je technika, při které dochází ke zpracování příchozích nebo odchozích dat přímo v hardware (na síťové kartě). Tato data by jinak musel zpracovávat procesor. Díky offloadingu se zmenšuje zátěž na CPU. Offloadů existuje mnoho různých typů, přičemž síťová karta může podporovat pouze některé z nich. Mezi nejčastější typy offloadů patří výpočty kontrolních součtů. Nastavení offloadů lze vypsat pomocí příkazu `ethtool -k`.

Název offloadu:	**Stav**
`rx-checksumming:`	zapnuto
`tx-checksumming:`	zapnuto
`tx-checksum-ip-generic:`	zapnuto
`tcp-segmentation-offload:`	vypnuto
`tx-tcp-segmentation:`	zapnuto
`tx-tcp-ecn-segmentation:`	vypnuto
`tx-tcp6-segmentation:`	zapnuto
`udp-fragmentation-offload:`	vypnuto (fixní hodnota)
`generic-segmentation-offload:`	zapnuto
`generic-receive-offload:`	zapnuto
`large-receive-offload:`	vypnuto

Virtuální host byl vždy spouštěn prostřednictvím emulátoru Qemu. Virtuálnímu hostovi bylo přiřazeno 8 GiB operační paměti a 3 virtuální CPU jádra, která byla namapována na *3 jádra fyzická* (v poměru 1:1). Komunikace s virtuálním hostem vždy probíhala prostřednictvím paravirtualizovaných síťových rozhraní (ovladač virtio). Nastavení offloadů těchto rozhraní lze vidět v následující tabulce:

Název offloadu:	**Stav**
`rx-checksumming:`	vypnuto
`tx-checksumming:`	zapnuto
`tx-checksum-ip-generic:`	zapnuto
`tcp-segmentation-offload:`	vypnuto
`tx-tcp-segmentation:`	vypnuto
`tx-tcp-ecn-segmentation:`	vypnuto
`tx-tcp6-segmentation:`	vypnuto
`udp-fragmentation-offload:`	zapnuto
`generic-segmentation-offload:`	zapnuto
`generic-receive-offload:`	zapnuto
`large-receive-offload:`	vypnuto

V případě topologie PVP docházelo k přeposílání vždy až na úrovni virtuálního hosta - v Linux-Bridge (2.2.2). Vytvoření virtuálního přepínače v hostovi lze vidět v ukázce 6.1.

```
# Povolí v linuxovém jádře forwarding
sysctl -w net.ipv4.ip_forward=1

# Vytvoření virtuálního přepínače br0
brctl addbr br0

# Přidání rozhraní eth0 a eth1 do br0
brctl addif br0 eth0
brctl addif br0 eth1

# Povolení přepínání na br0
ip link set dev br0 up
```

Kód 6.1: Ukázka konfigurace přepínače Linux-Bridge.

Nastavení OpenFlow pravidel v přepínači Open vSwitch pro vytvoření topologie PVP lze nalézt v příloze D v ukázce D.1.

6.2.2 Nastavení systému při testování s knihovnou DPDK

Nastavení systému při testování s knihovnou DPDK je náročnější. Systém DUT musel být vždy zaveden s podporou pro jednotku IOMMU (viz 3.1.1). Na DUT bylo vždy alokováno 16 1GiB velkých stránek (8 velkých stránek na každém NUMA uzlu). Při testování topologie PVP byly na virtuálním hostovi vždy alokovány 4 1GiB velké stránky. Síťové karty 82599ES byly pomocí `dpdk-devbind` odebrány ze správy modulu `ixgbe` a byl jim nastaven nízkoúrovňový ovladač `vfio-pci` (viz 3.1.2). Při testování topologie PVP musel být modul vfio-pci zaveden i ve virtuálním hostovi (tentokrát ale s vypnutou podporou pro IOMMU - volba `enable_unsafe_noiommu_mode=Y`). Následně i virtuální rozhraní hosta musela být převedena pod správu tohoto nízkoúrovňového modulu.

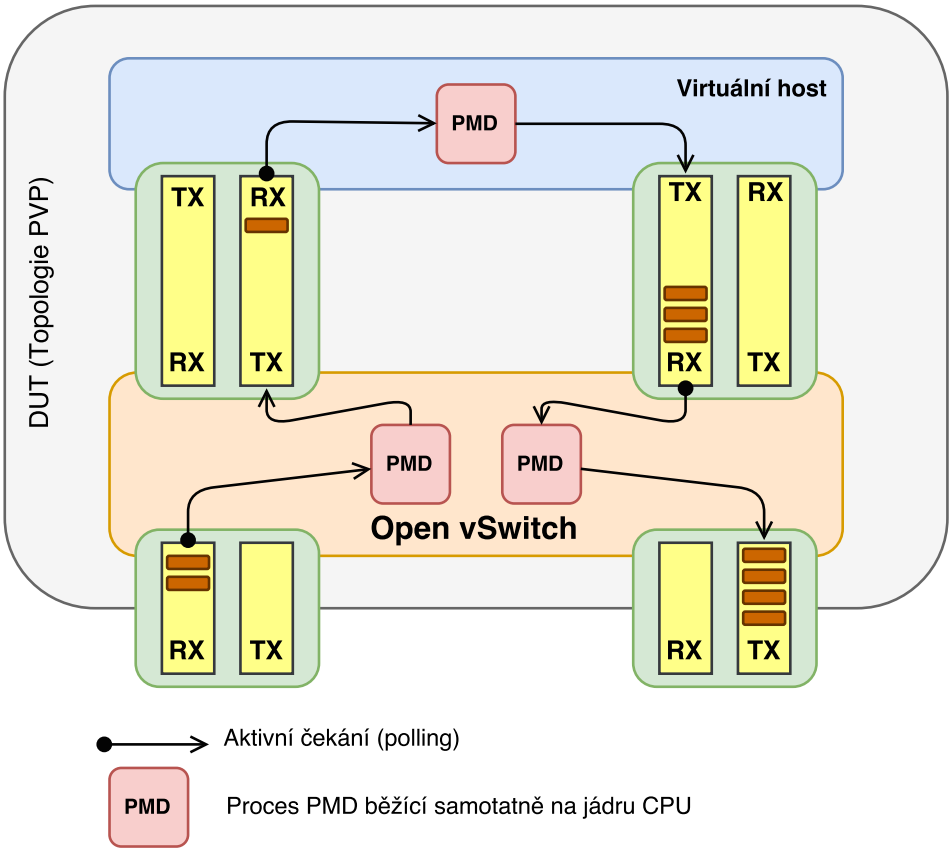
Jako další musela být na DUT i na virtuálním hostovi provedena izolace CPU a přerušení (viz sekce 3.1.1 a 5.3.1).

Je nutné zdůraznit, že polling prováděný vláknem PMD je vždy prováděn nad RX frontami. Na obrázku 6.1 lze vidět komunikaci znázorněnou graficky. Obrázek ukazuje jak probíhá polling pouze *v jednom směru komunikace*. Pro druhý směr probíhá vše symetricky. Celkově je tedy nutné mít pro topologii PVP k dispozici šest CPU jader, které budou provádět polling.

Mapování zátěže na jednotlivá jádra CPU u topologie P2P lze vidět na obrázku 6.2. Mapování jader pro topologii PVP je znázorněno na obrázku 6.3. Jak lze vidět, celé testování probíhá pouze na jednom soketu - na tom, ke kterému je připojena 10Gbit síťová karta. Druhý soket se testování vůbec neúčastní, protože by zde docházelo ke kopírování dat mezi NUMA uzly[2]. To by na výsledky testů pravděpodobně mělo špatný vliv.

Pokud komunikace probíhala přes virtuálního hosta, docházelo k přeposílání pomocí nástroje `testpmd`. Testpmd je DPDK aplikace, která je elementární implementací L2 přepínače. Tento program je přímou součástí repozitáře knihovny DPDK. V příkladu 6.2 lze vidět jak je testpmd spouštěn.

[2]Taková topologie se jmenuje PVVP. I tu testy implementované v rámci této práce podporují. Její detailní popis by byl ale nad rámec této práce. Více v sekci 4.2.



Obrázek 6.1: Polling nad RX frontami v jednom směru komunikace u topologie PVP.

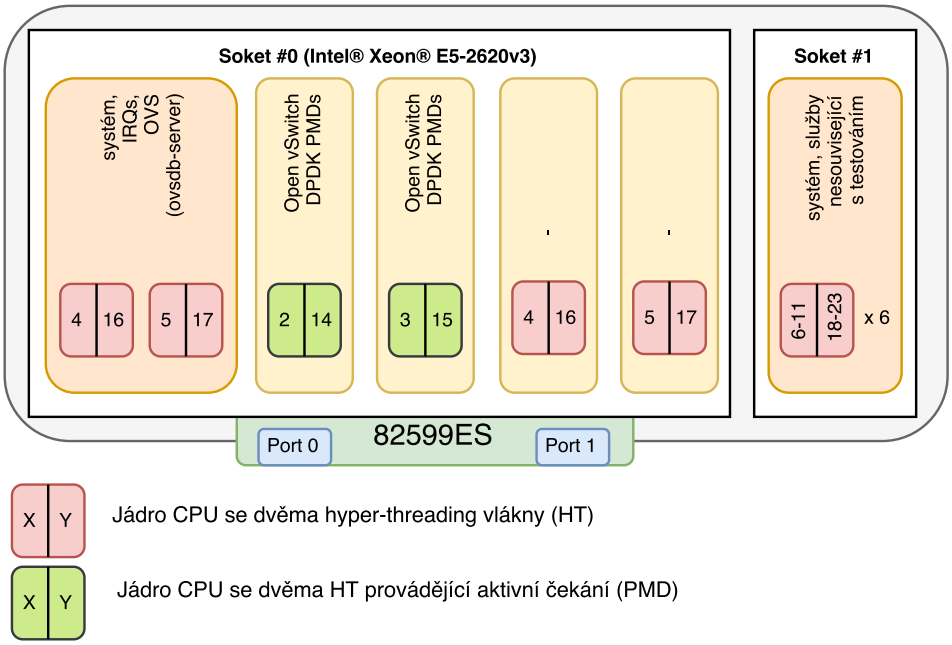
```
testpmd -l 0,1,2 \                # Specifikuje jádra, na kterých bude testpmd
                                  # běžet (0 - master vlákno, 1 a~2 - PMD vlákna).
        -n 4 \                    # Počet paměťových kanálů.
        --socket-mem 512 -- \     # Paměť alokovaná na daném soketu (v~bajtech).
        --burst=64 \              # Počet paketů přeposlaných v~jedné várce.
        -i - \                    # Zapne interaktivní mód.
        -txqflags=0xf00 \         # Maska specifikující parametry TX fronty.
        --disable-hw-vlan         # Vypne HW podporu pro VLAN.
testpmd> set fwd csum             # Nastaví mód přeposílání.
testpmd> start                    # Aktivuje přeposílání.
```

Kód 6.2: Ukázka spuštění programu testpmd.

Nastavení OpenFlow pravidel v přepínači Open vSwitch pro vytvoření topologie PVP s podporou DPDK lze vidět v příloze ve výpisu D.2.

6.3 Změření výkonu Open vSwitch

Pro potřeby diplomové práce byly obě topologie (4.2) nakonfigurovány a byla změřena jejich propustnost jak s použitím knihovny DPDK, tak i bez ní. Všechny testy byly prováděny na základě metodologie popsané v kapitole 4.



Obrázek 6.2: Rozložení zátěže pro jednotlivá CPU u topologie P2P.

Jako generátor provozu byl použit MoonGen (5.1.5). Důvodem jeho výběru byla především jeho dobrá podpora v programu VSperf (5.1.4), snadná instalace a jeho jednoduchá rozšiřitelnost o další funkce v budoucnosti. Tím, že se jedná o softwarový generátor, lze ho použít na relativně velkém množství serverů podporujících DPDK. Není tak potřeba specializovaný jednoúčelový hardwarový generátor, jehož cena se v případě vyšších rychlostí pohybuje v řádu desetitisíců dolarů[3].

Pro měření byly zvoleny čtyři různé velikosti rámců - 64, 576, 1280 a 1500 bajtů. Větší velikosti rámců (tzv. jumbo frames) nebyly testovány, protože jejich generování generátor MoonGen zatím nepodporuje.

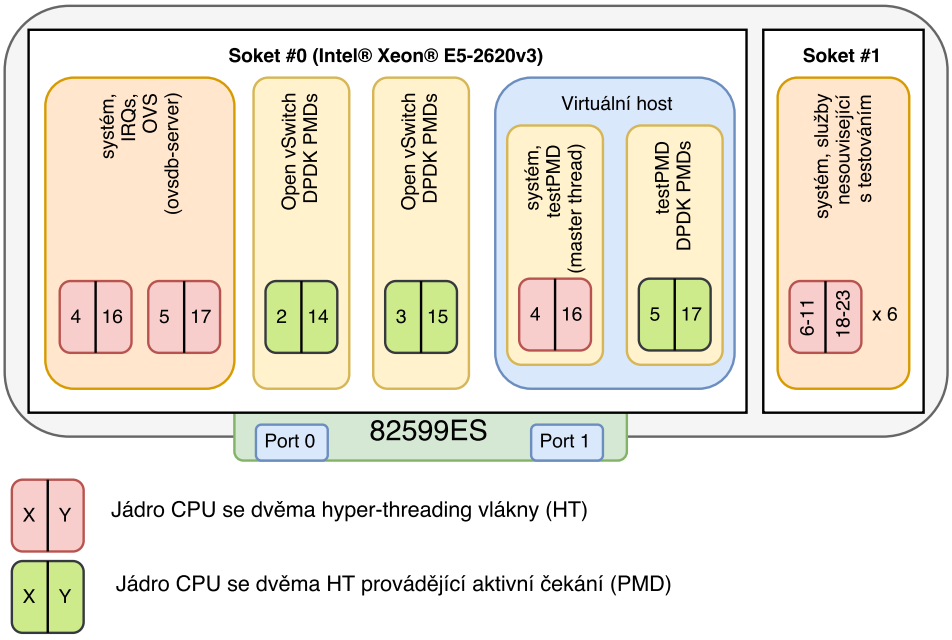
U všech měření prováděných s knihovnou DPDK byla nastavena maximální ztrátovost rámců na nulovou (žádný z odeslaných rámců se nesměl ztratit). Měření uskutečněná bez DPDK (čistě s využitím linuxového jádra) byla prováděna se ztrátovostí 0.001%, a to především z toho důvodu, že výsledky pro nulovou ztrátovost byly velmi nestabilní.

Pozn.: Ve všech následujících grafech je propustnost znázorněna v jednotkách Mbit/s jako ***součet rychlostí v obou směrech*** (více viz 4.1.1). Všechny tyto grafy byly vygenerovány manuálně. Na to, jak vypadá automatizované porovnání výsledku s baseline, se lze podívat v příloze E.

6.3.1 Výkon Open vSwitch

Jako první byl přepínač Open vSwitch proměřen bez jakéhokoliv zásahu knihovny DPDK. Byl otestován především modul linuxového jádra implementující jeho datovou vrstvu (2.2.4).

[3]Ceny těchto zařízení bohužel nejsou veřejně dostupné. Cena hardwarového generátoru provozu byla poptávána interně v rámci firmy Red Hat.



Obrázek 6.3: Rozložení zátěže pro jednotlivá CPU u topologie PVP.

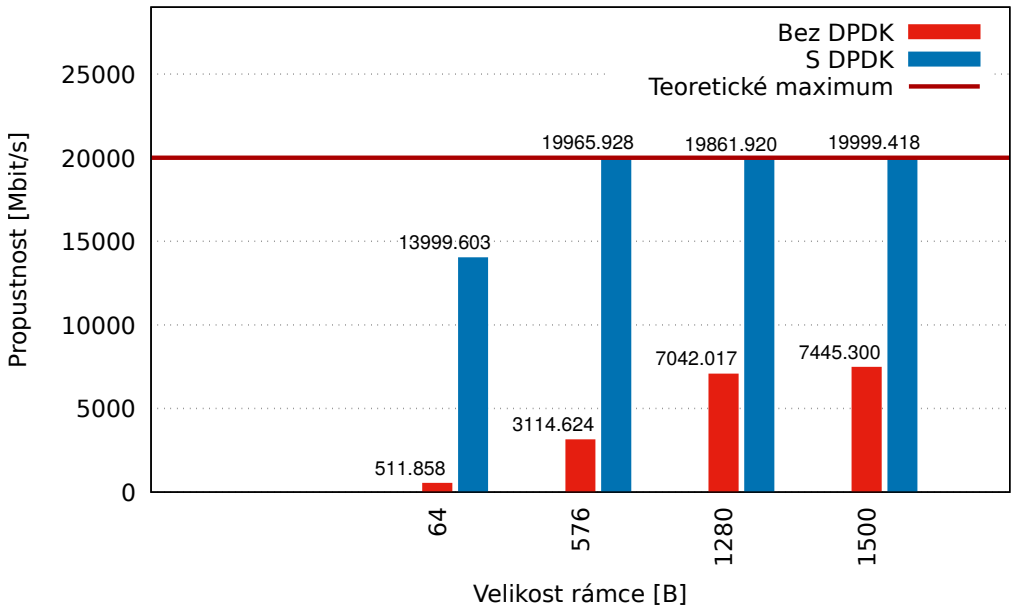
Zde je dobré poznamenat, že při testování docházelo ke kompletnímu zahlcení CPU jádra zpracovávajícího přerušení. Proces `irqbalance` zabíral 100% procesorového času.

Topologie P2P

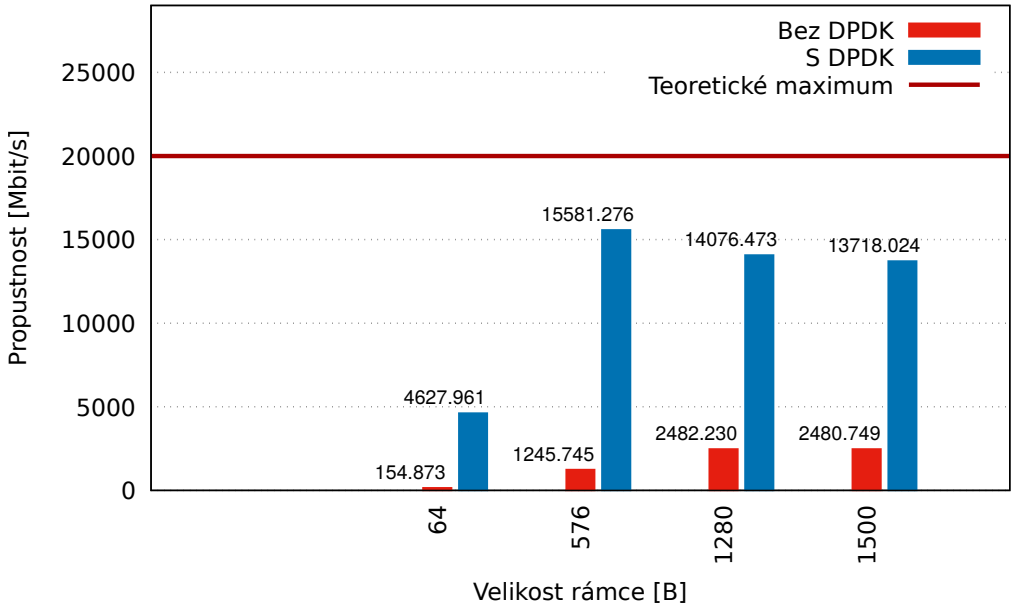
Jak lze vidět na obrázku 6.4, propustnost systému DUT při přeposílání extrémně malých rámců je bez DPDK (s využitím přerušení) velmi nízká (červené hodnoty v grafu). Propustnost systému v případě využití knihovny DPDK je výrazně lepší. Jak lze vidět, plný potenciál 10Gbit linky lze u topologie P2P plně využít už u rámců o velikosti 576 bajtů.

Topologie PVP

Při přeposílání malých rámců až na úrovni virtuálního hosta v Linux-Bridge nesahá hodnota propustnosti ani na pouhých **200 Mbit/s**. Výstup testu lze vidět na obrázku 6.5.



Obrázek 6.4: Měření propustnosti topologie P2P (bez DPDK).



Obrázek 6.5: Měření propustnosti PVP (bez DPDK).

Kapitola 7

Zhodnocení výsledků

Při prvním pohledu na výsledky měření v kapitole 6 by se mohlo zdát, že využití DPDK pro komunikaci se síťovým adaptérem je velmi výhodné. Proč tedy nepoužít DPDK akceleraci, když dává o tolik lepší výsledky?

Myšlenka DPDK spočívá v tom, že DPDK aplikace a vlákna PMD *běží stále* a naplno vytěžují jim přiřazená CPU jádra. Použít knihovnu DPDK je vhodné pouze v určitých případech. Celý koncept DPDK s sebou přináší nové problémy jako například:

- **Velkou spotřebu CPU jader** – Každé PMD vlákno provádějící polling musí mít vyhrazené jádro CPU. V případě topologie P2P musela být jen pro polling vyhrazena dvě jádra. U topologie PVP jich již muselo být šest. Kdyby pak na systému běžela složitější virtuální funkce, byla by jádra potřeba ještě pro ni. Další jádra by byla zapotřebí při využití technologie multiqueue (5.3.1). U té by byl pro podporu dvou RX a dvou TX front zapotřebí *dvojnásobný počet jader*.
- **Větší výdaje za spotřebovanou energii** – Vysoký počet CPU jader spotřebovávajících 100% procesorového času má za následek vyšší náklady spojené s provozem celého systému.
- **Větší náklady spojené s vývojem DPDK aplikace** – Jak již bylo řečeno, zpracování vyšších vrstev než L2 musí být dodatečně implementované přímo v uživatelském prostoru. Implementovat správně fungující DPDK aplikaci na tak nízké úrovni není jednoduché.

Použití knihovny DPDK musí být pevně zváženo. V případě, že virtuální síťová funkce nemá vysoké nároky na propustnost, latenci a ve většině případů nebude pracovat s extrémně malými rámci, plně dostačuje použití řešení bez DPDK. Implementace síťové funkce bez DPDK je o mnoho jednodušší hlavně z toho důvodu, že může využívat již existujících dlouhodobě otestovaných řešení - především síťové vrstvy linuxového jádra.

7.1 Optimální režim Open vSwitch s DPDK pro různé aplikace

Optimální režim konfigurace přepínače Open vSwitch je přímo závislý na jeho případu užití. Vždy je nutné zvážit, co přesně se od přepínače očekává.

7.1.1 Open vSwitch pouze jako přepínač

V tomto případu užití neplní Open vSwitch žádnou jinou funkci než přepínání mezi fyzickými rozhraními. Nevyužívá zde funkce žádné jiné DPDK aplikace. V tomto režimu v podstatě pouze nahrazuje hardwarový přepínač.

Jaký smysl tedy dává použití Open vSwitch v tomto režimu, když je možné použít hardwarový fyzický přepínač? Důvodem je především znovupoužitelnost a flexibilita ve využití hardware. Hardware, na kterém virtuální přepínač běží, je kompletně znovupoužitelný. V průběhu času může sloužit k absolutně jiným účelům než k přepínání. Znovupoužitelnost hardware platí i u všech následujících případů užití.

Další velkou výhodou je konfigurovatelnost přepínače pomocí protokolu OpenFlow. Díky tomu je možné přepínač použít při návrhu SDN sítí.

Optimálním režimem pro daný případ užití je následující konfigurace přepínače. Na systému je nutné vyhradit jádro CPU pro každé síťové rozhraní zúčastňující se přepínání. Pro rozhraní využívající technologie multiqueue je nutné vyhradit CPU jádro pro každou přijímající (RX) frontu. Toto jsou pouze jádra vyhrazená pro PMD provádějící polling. Další CPU jádra jsou potřeba pro logiku přepínání (především pro programy `ovs-vswitchd` a `ovsdb-server`) a pro běh samotného systému. Tyto jádra ale už mohou být sdílená více procesy.

Tomuto režimu odpovídá topologie P2P. Více o možnostech její konfigurace lze najít v sekcích 4.2.1 a 6.2.2.

Tento případ užití sice nedisponuje výhodami virtualizovaného prostředí, ale výkonnostně dosahuje daleko lepších výsledků než režim třetí (7.1.3). Kapacitu 10Gbit linky lze při rámcích o velikosti 64 bajtů využít ze 70% (viz 6.3).

Pozn. k technologii multiqueue: Celkový výkon přepínače se při použití technologie multiqueue zvýší, nebude se ale s každým přidaným CPU jádrem zvyšovat lineárně. Přidání dalšího CPU jádra *nezaručí dvojnásobné navýšení výkonu.*

7.1.2 DPDK aplikace přímo nad Open vSwitch

V tomto režimu funguje přepínač stejně jako v režimu předchozím (7.1.1), ale navíc zde využívá funkcí ještě jiné DPDK aplikace (tzv. síťové funkce). Ta běží přímo nad přepínačem (neběží ve virtuálním hostovi). Do ní je přepínač schopný odesílat přesně specifikované toky dat. DPDK aplikace pak nad obdrženými daty provádí různé operace. Může provádět například funkci filtrování, směrování, odstranění/přidání VLAN značení, zajišťovat QoS a mnoho dalšího.

Optimální režim fungování tohoto případu užití je velmi podobný režimu předchozímu. Navíc je zde ale nutné přidat CPU jádra i pro každé *virtuální rozhraní přepínače* (přesněji pro každou RX frontu rozhraní). Zároveň je potřeba počítat i jádra provádějící síťovou funkci. DPDK aplikace provádějící síťovou funkci vyžaduje nejméně dvě jádra CPU. Jedno z nich musí být dedikované pro polling pomocí PMD. Druhé z nich může být sdílené s jinými procesy. Na něm poběží samotná aplikace provádějící síťovou funkci.

Tento režim použití dává smysl v případě, že fyzický host nedisponuje velkým množstvím CPU jader. Pokud má fyzický systém CPU jader dostatek, je výhodnější použít režim s využitím virtualizace (viz dále).

7.1.3 DPDK aplikace ve virtualizovaném prostředí

Posledním zmíněným případem užití je rozšíření předchozího (7.1.2) o výhody virtualizace. DPDK aplikace, v tomto kontextu zvaná také jako virtuální síťová funkce, běží ve virtuálním hostovi kompletně odděleně od hostujícího systému.

Virtualizace s sebou přináší mnoho výhod. Mezi nejzajímavější určitě patří:

- Možnost uložení stavu systému v určitém časovém okamžiku (tzv. snímkování).
- Možnost přesunutí (tzv. migrace) virtuálních hostů mezi hypervizory.
- Nezávislost virtuální síťové funkce na hardware.
- Vyšší bezpečnost.
- Možnost delegovat správu jednotlivých virtuálních hostů (a tedy i virtuálních síťových funkcí) na různé administrátory.

Optimální režim přepínače Open vSwitch je opět velmi podobný předchozím. Je zde opět potřeba CPU jádro pro každé rozhraní přepínače (přesněji pro každou RX frontu). Dále jsou potřeba minimálně dvě izolovaná CPU jádra pro běh DPDK aplikace ve virtualizovaném prostředí. Na prvním z jader poběží polling ovladače PMD. Na druhém jádře poběží DPDK aplikace zároveň s operačním systémem virtuálního hosta. Kdyby DPDK aplikaci sdílené jádro nestačilo, je pro ni možné vyhradit jádro dedikované.

Tento případ užití úzce koresponduje s topologií PVP. Více o možnostech její konfigurace lze najít v sekcích 4.2.2 a 6.2.2.

Daní za větší flexibilitu celého řešení je bohužel výkon. Ten je oproti předchozím zmíněným režimům nižší. Propustnost se ale stále pohybuje v řádu *jednotek gigabitů za sekundu*. Více o výsledcích měření přepínače lze najít v sekci 6.3.

Kapitola 8

Závěr

Práce popisuje nejnovější trendy v oblasti virtualizace různých síťových zařízení (virtuálních funkcí). Nejvíce se ale zaměřuje na tématiku virtuálních přepínačů - především jejich akcelerace.

Text této práce se zabývá otevřeným virtuálním přepínačem Open vSwitch. Přepínač Open vSwitch byl v práci rozebrán a byla vysvětlena jeho architektura. Práce popisuje jeho vlastnosti a výhody jeho použití v SDN sítích. Zároveň byly krátce popsány i další virtuální přepínače, aby bylo zřejmé, jaké výhody přepínač Open vSwitch přináší (2).

Dále je zde ukázáno v jakých situacích výkon virtuálních přepínačů nedostačuje. Z toho důvodu se další kapitola práce (3) se zabývá možnostmi akcelerace přepínače Open vSwitch. Představuje technologie obcházející jadernou síťovou vrstvu. Zde se zabývá hlavně knihovnou DPDK. U té rozebírá její vlastnosti. Popisuje SW i HW technologie nutné pro její bezproblémový chod a dosažení maximálního výkonu. Zároveň detailně popisuje celou její architekturu a její jednotlivé funkční celky.

Znalosti načerpané při studiu problematiky v těchto dvou kapitolách byly využity pro návrh metodologie testování výkonu přepínače Open vSwitch jak s podporou knihovny DPDK, tak i bez ní. Celá metodologie testování propustnosti je založena na RFC2544. Jsou zde představeny dvě síťové topologie zapojení přepínače Open vSwitch (P2P a PVP) jejichž testováním se práce zabývá (4).

Nejdůležitější a největší část práce tvoří návrh a implementace plně automatizovaného prostředí (tzv. CI procesu) pro testování výkonu přepínače Open vSwitch (5). Celý proces automatizace využívá velkých automatizačních systémů, především Koji, Jenkins a Beaker. Pro samotné testování je použit generátor provozu MoonGen ve spojení s nástrojem VSperf. Nástroj VSperf umožňuje automatizovaně konfigurovat různé topologie přepínače Open vSwitch jak s podporou DPDK, tak i bez ní. Umožňuje také automatizovaně spustit generátor provozu. Pro konfiguraci a spuštění nástroje VSperf byly vytvořeny pomocné skripty, které celé testování abstrahují. Příkladem těchto skriptů je úloha `/performance/vsperf_test` a nástroj JobCreator (5.2.2).

Práce se zabývá i zpracováním získaných výsledků měření. K tomu byl navržen modul zvaný `reporting` (5.4), který výsledky měření automaticky porovnává s výsledky referenčními.

Na základě navržené testovací metodologie a implementovaného CI procesu testování bylo provedeno měření dvou zvolených topologií přepínače Open vSwitch na zvoleném hardware. Měření bylo provedeno jak s podporou knihovny DPDK, tak i bez ní (6).

Výsledky měření byly zhodnoceny (7). Bylo zjištěno, že s použitím knihovny DPDK se dá dosáhnout daleko lepších výkonnostních výsledků. Bohužel na úkor spotřebované energie

a využití velkého množství jader CPU. Tato jádra musí být totiž izolovaná a vyhrazená především pro účely aktivního čekání na příchozí data.

I přes tyto negativní vlastnosti existují případy, kdy použití přepínače Open vSwitch s DPDK dává smysl[1]. Tyto případy užití byly vysvětleny ve spojení s tzv. síťovými funkcemi. Pro každý zmíněný případ užití byl popsán optimální režim konfigurace přepínače Open vSwitch.

V rámci práce bylo přispěno i do několika komunitních projektů, především do projektu VSperf [37] a Lua skriptu pro generátor MoonGen implementujícího testování propustnosti dle RFC2544 (5.1.5).

Celé implementované automatizované prostředí je v rámci firmy Red Hat kompletně nasazeno a je používáno pro pravidelné testování přepínače Open vSwitch i knihovny DPDK.

8.1 Další možnosti rozšíření

- **Testování multiqueue** – Jakmile bude společnost Red Hat schopna vyhradit stroje s vyšším počtem jader, bude testovací prostředí rozšířeno o testování multiqueue. Testování multiqueue je v této práci zmíněno pouze okrajově. Je to z toho důvodu, že systémy, na kterých probíhalo testování, nedisponovaly dostatečným počtem jader na CPU (6.1). Testy samotné jsou na multiqueue připraveny.

- **Podpora více druhů síťových karet** – Testy budou v budoucnu pravděpodobně rozšířeny o možnost testování více druhů síťových adaptérů. Především adaptérů značek Solarflare (`sfc_efx`) a Broadcom (`bnxt`). Časem budou testy rozšířeny také o podporu 40Gbit karet. K tomu je ale nutný přechod na generátor provozu, který takové rychlosti podporuje (např. T-Rex).

- **Přechod na generátor provozu T-Rex** – V průběhu implementace automatizovaného prostředí bylo zjištěno, že generátor MoonGen je mnoha ohledech nedostačující. V budoucnu bude pravděpodobně nutné přejít na generátor jiný. T-Rex je jeden z možných kandidátů.

- **Měření latence** – Výkon přepínače Open vSwitch by neměl být založený pouze na výsledcích propustnosti. Je nutné znát i latenci s jakou data systémem DUT procházejí. Podpora pro měření latence je v generátoru MoonGen téměř hotova. Jakmile bude dokončena, bude v rámci implementovaných testů měřena i latence.

- **Testování různých jader OS** – Implementované testy budou rozšířeny o možnost testování různých verzí linuxových jader. I jádro totiž může mít vliv na celkový výkon testované topologie.

- **Vylepšení modulu reporting** – V modulu `reporting` je velký prostor pro možná rozšíření. V budoucnu by bylo více než žádoucí zobrazovat ve výsledném přehledu i spotřebované systémové zdroje a jejich porovnání s referenčním výsledkem.

[1] Důkazem použití DPDK v praxi může být firma 6WIND, která nabízí již existující řešení postavená nad knihovnou DPDK [1].

Literatura

[1] 6WIND: *6WINDGate*. Květen 2017, [Online; navštíveno 15.05.2017].
URL http://www.infoworld.com/article/2621446/server-virtualization/server-virtualization-top-10-benefits-of-server-virtualization.html

[2] Bovet, D. P.; Cesati, M.: *Understanding the Linux Kernel, Third Edition*. O'Reilly Media, Inc., 2016, ISBN 978-0596005658.

[3] Bradner, S.: *Benchmarking Terminology for Network Interconnection Devices*. RFC 1242, RFC Editor, Červenec 1991.
URL http://www.rfc-editor.org/rfc/rfc1242.txt

[4] Cisco: *Virtual Networking*. [Online; navštíveno 06.01.2017].
URL https://www.cisco.com/c/en/us/products/switches/virtual-networking/index.html

[5] Corbet, J.: *Improving Linux networking performance*. Leden 2015, [Online; navštíveno 08.01.2017].
URL https://lwn.net/Articles/629155

[6] Davis, D.: *Server Virtualization, Network Virtualization & Storage Virtualization Explained*. Leden 2009, [Online; navštíveno 06.01.2017].
URL https://www.petri.com/server-virtualization-network-virtualization-storage-virtualization

[7] Dittner, R.; Rule, D.: *The Best Damn Server Virtualization Book Period*. Syngress Publishing, 2007, ISBN 978-1-59749-217-1.

[8] Dražil, J.: *Optimalizace síťových úloh*. Vysoké učení technické v Brně, Fakulta informačních technologi, 2016, vedoucí práce Viktorin Jan.

[9] Emmerich, P.; Gallenmüller, S.; Raum, D.: *MoonGen: A Scriptable High-Speed Packet Generator*. Říjen 2015.

[10] Emmerich, P.; Raumer, D.; Wohlfart, F.; aj.: *Performance Characteristics of Virtual Switching*. 2014.

[11] García, D. P.: *Multicore architectures and CPU affinity*. Říjen 2015, [Online; navštíveno 08.01.2017].
URL http://blogs.igalia.com/dpino/2015/10/15/multicore-architectures-and-cpu-affinity/

[12] Gorman, M.: *Huge pages*. Únor 2010, [Online; navštíveno 08.01.2017].
URL https://lwn.net/Articles/374424

[13] Haldar, S.; Aravind, A.: *Operating Systems*. Pearson Education, Květen 2010, ISBN 978-8131730225.

[14] Intel: *Intel Virtualization Technology for Directed I/O - Architecture Specification*. Červen 2016.
URL https://www-ssl.intel.com/content/dam/www/public/us/en/documents/product-specifications/vt-directed-io-spec.pdf

[15] Kroah-Hartman, G.: *Manual driver binding and unbinding*. Červenec 2005, [Online; navštíveno 08.01.2017].
URL https://lwn.net/Articles/143397

[16] remote lab.net: *SDN Intro: Basic L2 connectivity by using OVS and POX*. [Online; navštíveno 09.01.2017].
URL https://remote-lab.net/sdn-intro-basic-with-ovs-and-pox

[17] Maintainers of DPDK.org: *DPDK Homepage*. [Online; navštíveno 08.01.2017].
URL http://dpdk.org

[18] Mandeville, R.; Perser, J.: *Benchmarking Methodology for LAN Switching Devices*. RFC 2889, RFC Editor, Srpen 2000.
URL http://www.rfc-editor.org/rfc/rfc2889.txt

[19] Marshall, D.: *Top 10 benefits of server virtualization*. Listopad 2011, [Online; navštíveno 06.01.2017].
URL http://www.infoworld.com/article/2621446/server-virtualization/server-virtualization-top-10-benefits-of-server-virtualization.html

[20] McQuaid, J.; Bradner, S.: *Benchmarking Methodology for Network Interconnect Devices*. RFC 2544, RFC Editor, Březen 1999.
URL http://www.rfc-editor.org/rfc/rfc2544.txt

[21] Pall, M.: *LuaJIT*. [Online; navštíveno 27.04.2017].
URL http://luajit.org/luajit.html

[22] Pettit, J.; Gross, J.; Pfaff, B.; aj.: *Virtual Switching in an Era of Advanced Edges*. 2010.

[23] Pfaff, B.; Davie, B.: *The Open vSwitch Database Management Protocol*. RFC 7047, RFC Editor, Prosinec 2013.
URL http://www.rfc-editor.org/rfc/rfc1747.txt

[24] Pfaff, B.; Pettit, J.; Koponen, T.; aj.: *The Design and Implementation of Open vSwitch*. Networked Systems Design and Implementation, Květen 2015, ISBN 978-1-59749-217-1.

[25] Rajaram, R.: *Presentation - Introduction to Open vSwitch (Presented on FUDCon Pune)*. Červen 2015, [Online; navštíveno 06.01.2017].
URL http://www.slideshare.net/rranjithrajaram/open-vswitch-49918658

[26] Rizzo, L.: *netmap: a novel framework for fast packet I/O*. [Online; navštíveno 08.01.2017].

[27] Rizzo, L.: *netmap - the fast packet I/O framework*. [Online; navštíveno 08.01.2017].
URL http://info.iet.unipi.it/~luigi/netmap

[28] SDX Central: *What is Open vSwitch*. Červen 2016, [Online; navštíveno 06.01.2017].
URL https://www.sdxcentral.com/cloud/open-source/definitions/what-is-open-vswitch

[29] Tahhan, M.: *Benchmarking Virtual Switches in OPNFV*. Technická zpráva, Květen 2017.
URL https://tools.ietf.org/html/draft-ietf-bmwg-vswitch-opnfv-03

[30] TRex Team: *TRex Realistic traffic generator*. [Online; navštíveno 09.01.2017].
URL https://trex-tgn.cisco.com

[31] Více autorů: *Beaker Project - Documentation*. [Online; navštíveno 17.04.2017].
URL https://beaker-project.org

[32] Více autorů: *Linux Kernel Documentation*. [Online; navštíveno 08.01.2017].
URL https://www.kernel.org/doc

[33] Více autorů: *Open vSwitch Database Schema*. [Online; navštíveno 06.01.2017].
URL http://openvswitch.org/ovs-vswitchd.conf.db.5.pdf

[34] Více autorů: *Open vSwitch Documentation*. [Online; navštíveno 06.01.2017].
URL http://docs.openvswitch.org/en/latest/contents/

[35] Více autorů: *Open vSwitch manpages*. [Online; navštíveno 06.01.2017].
URL http://openvswitch.org/support/dist-docs

[36] Více autorů: *OPNFV*. [Online; navštíveno 19.04.2017].
URL https://www.opnfv.org

[37] Více autorů: *OPNFV - VSperf*. [Online; navštíveno 19.04.2017].
URL https://wiki.opnfv.org/display/vsperf/

[38] Více autorů: *The Fast Data Project*. [Online; navštíveno 09.01.2017].
URL https://fd.io

[39] Více autorů: *Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges & Call for Action*. Říjen 2012.

[40] Willis, N.: *A BoF on kernel network performance*. Únor 2016, [Online; navštíveno 08.01.2017].
URL https://lwn.net/Articles/676806

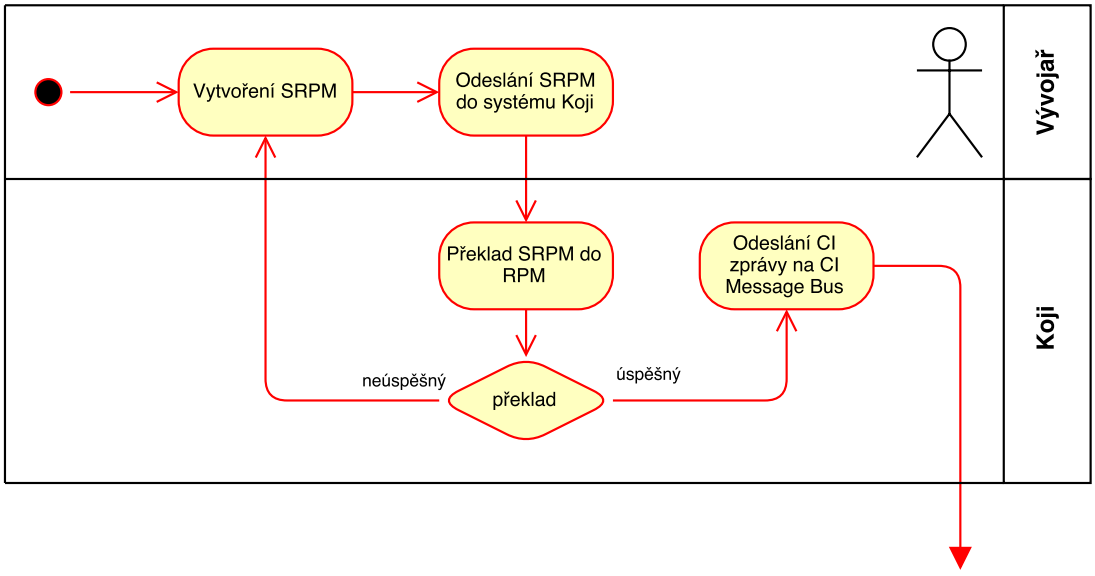
[41] Willmann, P.; Rixner, S.; Cox, A. L.: *Protection Strategies for Direct Access to Virtualized I/O Devices*. 2008.

Přílohy

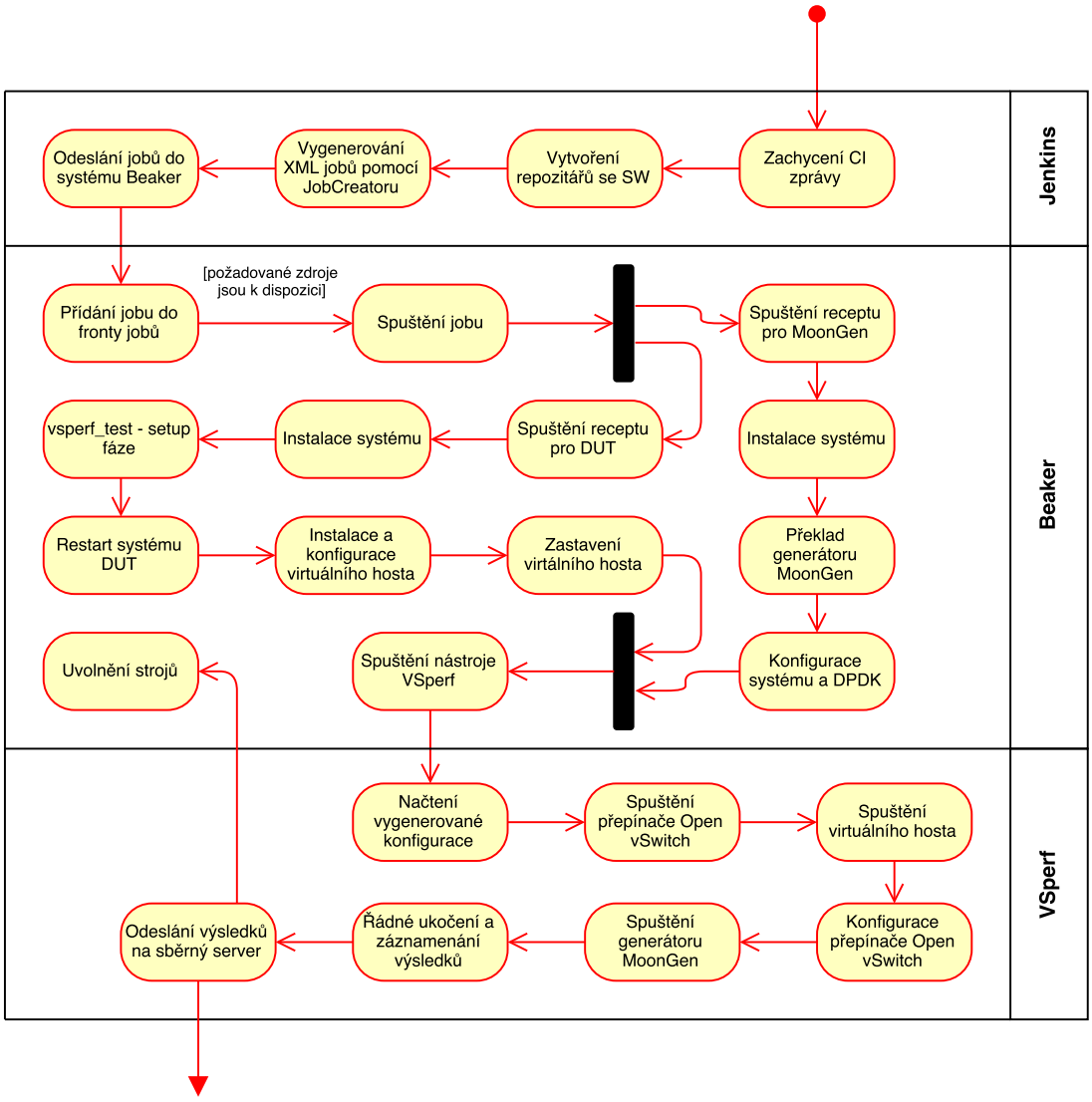
Příloha A

Grafické znázornění CI procesu

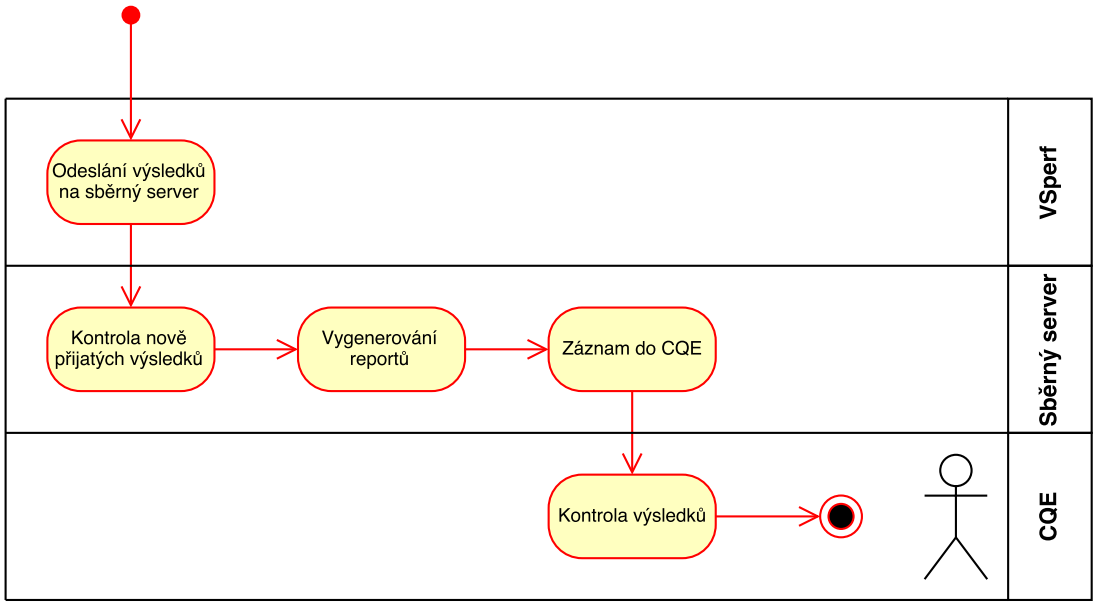
Diagram aktivit celého CI procesu musel být rozdělen do tří částí. První z nich (A.1) začíná od procesu vývoje a získání SW pro potřeby testování. Na druhém diagramu (A.2) je znázorněn proces testování za pomoci systému Jenkins, Beaker a VSperf. Na posledním (třetím) diagramu (A.3) je znázorněno zpracování výsledků na sběrném serveru pomocí modulu `reporting` a následná kontrola v systému CQE.



Obrázek A.1: Diagram aktivit CI procesu - příprava software (1).



Obrázek A.2: Diagram aktivit CI procesu - testování (2).



Obrázek A.3: Diagram aktivit CI procesu - zpracování výsledků (3).

Příloha B

Příklad Beaker jobu

XML kódy B.1 a B.2 musely být zjednodušeny z důvodu jejich délky.

```
<recipe kernel_options_post="intel_iommu=on iommu=pt default_hugepagesz=1G
    hugepagesz=1G hugepages=16">
  <packages>...</packages>
  <distroRequires>...</distroRequires>
  <hostRequires>...</hostRequires>
  <task name="/distribution/install"/>
  <task name="/distribution/crashes/enable-abrt"/>
  <task name="/performance/build_moongen"/>
  <task name="/performance/setup_dpdk">...</task>
  <task name="/distribution/utils/reboot"/>
  <task name="/performance/setup_dpdk">...</task>
  <task name="/performance/perf_synchronization">
    <params>
      <param name="SYNC_ARGS" value="--set MOONGEN_READY"/>
    </params>
  </task>
  <task name="/performance/perf_synchronization">
   <params>
     <param name="SYNC_ARGS" value="--wait VSPERF_END"/>
   </params>
  </task>
</recipe>
```

Kód B.1: Recept popisující konfiguraci generátoru provozu MoonGen.

```
<recipe kernel_options_post="intel_iommu=on iommu=pt default_hugepagesz=1G
    hugepagesz=1G hugepages=16">
  <guestrecipe guestargs="--ram=8192 --vcpus=4 --file-size=8 --hvm --kvm">
    <packages>...</packages>
    <distroRequires>...</distroRequires>
    <hostRequires>...</hostRequires>
    <task name="/distribution/install"/>
    <task name="/performance/vsperf_test">
      <params>
        <param name="RUN_VSPERF_ARGS" value="setup --guest
            OVS_DPDK1q_GUEST_DPDK1q"/>
      </params>
    </task>
  </guestrecipe>

  <packages>...</packages>
  <distroRequires>...</distroRequires>
  <hostRequires>...</hostRequires>

  <task name="/distribution/install"/>
  <task name="/distribution/crashes/enable-abrt"/>
  <task name="/distribution/virt/install"/>
  <task name="/performance/vsperf_test">
    <params>
      <param name="RUN_VSPERF_ARGS" value="setup OVS_DPDK1q_GUEST_DPDK1q"/>
    </params>
  </task>
  <task name="/distribution/utils/reboot"/>
  <task name="/performance/perf_synchronization">
    <params>
      <param name="SYNC_ARGS" value="--wait MOONGEN_READY"/>
    </params>
  </task>
  <task name="/distribution/virt/start"/>
  <task name="/distribution/virt/stop"/>
  <task name="/performance/vsperf_test">
    <params>
      <param name="RUN_VSPERF_ARGS" value="run OVS_DPDK1q_GUEST_DPDK1q"/>
    </params>
  </task>
  <task name="/performance/perf_synchronization">
    <params>
      <param name="SYNC_ARGS" value="--set VSPERF_END"/>
    </params>
  </task>
</recipe>
```

Kód B.2: Recept popisující konfiguraci DUT (s podporou DPDK).

Příloha C

Příklad konfiguračního souboru pro program VSperf

V ukázce C.1 lze vidět příklad konfiguračního souboru používaného pro testování scénáře `OVS_DPDK1q_GUEST_DPDK1q`. Příklad ukazuje pouze nejdůležitější nastavení.

```
# Velikosti testovaných rámců
TRAFFICGEN_PKT_SIZES = (64, 576, 1280, 1500, )

# Typ VNF
VNF = "QemuDpdkVhostUser"
VSWITCH = "OvsDpdkVhost"

# Typ generátoru provozu použitého k testování
TRAFFICGEN = "Moongen"

# Konfigurace generátoru
TRAFFICGEN_MOONGEN_BASE_DIR = "/mnt/build_moongen/lua-trafficgen"
TRAFFICGEN_MOONGEN_PORTS = "{0,1}"
TRAFFICGEN_MOONGEN_LINE_SPEED_GBPS = "10"

# Jako obraz virtuálního hosta se použije obraz vygenerovaný systémem Beaker
GUEST_IMAGE = ["/var/lib/libvirt/images/guest.img"]

GUEST_DPDK_BIND_DRIVER = ["vfio_no_iommu"]
GUEST_USERNAME = ["root"]
GUEST_PASSWORD = ["root"]
GUEST_MEMORY = ["8192"]
GUEST_HUGEPAGES_NR = ["4"]
GUEST_NICS = [
    [
        {
            "device" : "eth0",
            "mac" : "#MAC(00:00:00:00:00:01,2)",
            "pci" : "0000:00:03.0",
            "ip" : "#IP(192.168.1.2,4)/24"
        },
        {
            "device" : "eth1",
            "mac" : "#MAC(00:00:00:00:00:02,2)",
            "pci" : "0000:00:04.0",
            "ip" : "#IP(192.168.1.3,4)/24"
        },
    ]
]
```

Kód C.1: Konfigurační soubor pro nástroj VSperf.

Příloha D

Ukázka konfigurace přepínače Open vSwitch

D.1 Konfigurace topologie PVP bez DPDK

Příkazy v ukázce D.1 předpokládají, že všechny komponenty přepínače Open vSwitch běží a databáze přepínače je vytvořena. Postup pro vytvoření topologie P2P by byl obdobný.

```
# Vytvoření virtuálního přepínače jménem br0
ovs-vsctl add-br br0

# Přidání fyzických rozhraní do br0
ovs-vsctl add-port br0 ens1f0
ovs-vsctl add-port br0 ens1f1

# Přidání rozhraní virtuálního hosta do br0
ip tuntap add tap0 mode tap && ip link set dev tap0 up
ip tuntap add tap1 mode tap && ip link set dev tap1 up
ovs-vsctl add-port br0 tap0
ovs-vsctl add-port br0 tap1

# Smazání existujících OpenFlow pravidel
ovs-ofctl -O OpenFlow13 del-flows br0

# Přeposílání všech dat z rozhraní 1 na rozhraní 3 a obráceně
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=1,action=output:3
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=3,action=output:1

# Přeposílání všech dat z rozhraní 4 na rozhraní 2 a obráceně
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=4,action=output:2
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=2,action=output:4

# Virtuální rozhraní (tap0 a tap1) jsou předány emulátoru pomocí parametru
# při jeho spuštění. Příklad spuštění emulátoru byl pro přehlednost zkrácen.
qemu-system-x86_64 ... \
  -netdev type=tap,id=eth0,ifname=tap0,vhost=on \
  -device virtio-net-pci,mac=00:00:00:00:00:01,netdev=eth0 \
  ...
  -netdev type=tap,id=eth1,ifname=tap1,vhost=on \
  -device virtio-net-pci,mac=00:00:00:00:00:02,netdev=eth1 \
  ...
```

Kód D.1: Nastavení OpenFlow pravidel pro vytvoření topologie PVP.

D.2 Konfigurace topologie PVP s využitím DPDK

Příkazy v ukázce D.2 počítají s tím, že fyzická rozhraní `0000:06:00.0` a `0000:06:00.1` jsou ve správě modulu `vfio-pci`. Dále musí být spuštěny všechny komponenty přepínače a jeho databáze musí být předem vytvořena. Postup pro vytvoření topologie P2P by byl obdobný.

```
# Vytvoření virtuálního přepínače jménem br0
ovs-vsctl add-br br0 -- set bridge br0 datapath_type=netdev

# Inicializace knihovny DPDK
ovs-vsctl set Open_vSwitch . other_config:dpdk-socket-mem=1024,0
ovs-vsctl set Open_vSwitch . other_config:dpdk-init=true
ovs-vsctl set Open_vSwitch . other_config:dpdk-lcore-mask=0x002000
ovs-vsctl set Open_vSwitch . other_config:max-idle=30000
ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=00c00c

# Přidání rozhraní specifikovaných dle jejich PCI identifikátoru do br0
ovs-vsctl add-port br0 dpdk0 -- set Interface dpdk0 type=dpdk options:
    dpdk-devargs=0000:06:00.0
ovs-vsctl add-port br0 dpdk1 -- set Interface dpdk1 type=dpdk options:
    dpdk-devargs=0000:06:00.1

# Vytvoření virtuálních DPDK rozhraní (typu vhost-user) pro virtuálního hosta
# a jejich přidání do br0
ovs-vsctl add-port br0 dpdkvhostuser0 -- set Interface dpdkvhostuser0 type=
    dpdkvhostuser
ovs-vsctl add-port br0 dpdkvhostuser1 -- set Interface dpdkvhostuser1 type=
    dpdkvhostuser

# Smazání existujících OpenFlow pravidel
ovs-ofctl -O OpenFlow13 del-flows br0

# Přeposílání všech dat z rozhraní 1 na rozhraní 3 a obráceně
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=1,action=output:3
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=3,action=output:1

# Přeposílání všech dat z rozhraní 4 na rozhraní 2 a obráceně
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=4,action=output:2
ovs-ofctl -O OpenFlow13 add-flow br0 in_port=2,action=output:4

# Virtuální rozhraní (dpdkvhostuser0 a dpdkvhostuser1) jsou předány emulátoru
# pomocí parametru při jeho spuštění. Příklad spuštění emulátoru byl pro
# přehlednost zkrácen.
qemu-system-x86_64 ... \
  -chardev socket,id=char0,path=/var/run/openvswitch/dpdkvhostuser0 \
  -netdev type=vhost-user,id=net1,chardev=char0,vhostforce \
  -device virtio-net-pci,mac=00:00:00:00:00:01,netdev=net1 \
  ...
  -chardev socket,id=char1,path=/var/run/openvswitch/dpdkvhostuser1 \
  -netdev type=vhost-user,id=net2,chardev=char1,vhostforce \
  -device virtio-net-pci,mac=00:00:00:00:00:02,netdev=net2 \
  ...
```

Kód D.2: Nastavení OpenFlow pravidel pro vytvoření topologie PVP s DPDK.

Příloha E

Příklad výsledného HTML reportu

Report musel být kvůli jeho velikosti rozdělen do několika obrázků. Na obrázku E.1 lze v horní části vidět tabulku se souhrnem porovnávaných verzí přepínače Open vSwitch i knihovny DPDK, dobou testování a dva odkazy. První odkaz (`VSperf Report`) vede na HTML report vygenerovaný programem VSperf. Tento report obsahuje mnoho dalších detailů a systémových statistik z průběhu testování. Druhý odkaz vede na kompletní logy vygenerované úlohou `/performance/vsperf_test`. V další tabulce se už nachází porovnání propustnosti topologie P2P, v tomto případě pro dvě různé verze knihovny DPDK. Zde je znovu důležité zmínit, že hodnota propustnosti je vyjádřena jako součet propustností v obou směrech (více viz 6.3).

Na obrázku E.2 lze vidět stejné porovnání propustnosti, ale pro topologii PVP.

Celkový report ještě obsahuje tabulku, která přehledně zobrazuje základní informace o distribuci, kernelu a hardware, na kterém probíhalo testování.

Results overview

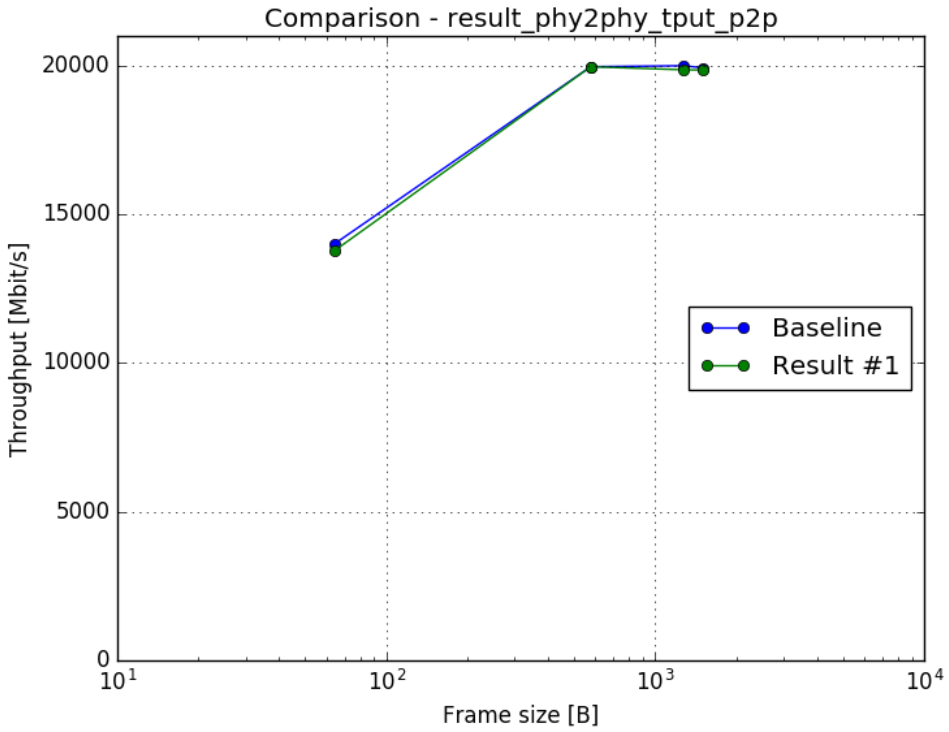
Because all performed tests are bi-directional, the result is a *sum* of throughputs in both directions.

Comparisons between:

Result ID	Open vSwitch	DPDK	Test time (rel to BS)	VSperf Report	Test log
Baseline	openvswitch-2.6.1-3.git20161206.el7fdb	dpdk-16.11-2.el7fdb	14.0 hours	report	logs
Result #1	openvswitch-2.6.1-13.git20161206.el7fdp	dpdk-16.11-4.el7fdb	took 0.0 hours	report	logs

result_phy2phy_tput_p2p

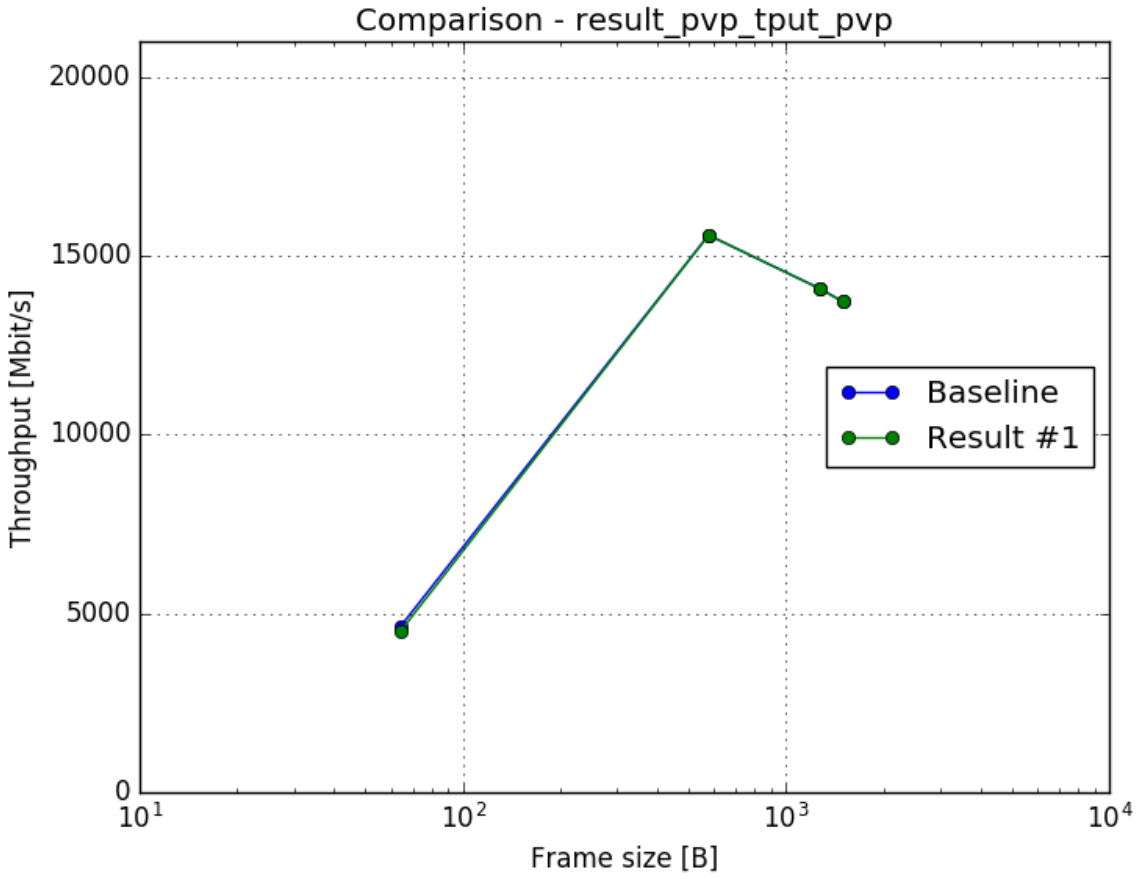
Result ID	Frame size	64 B		576 B		1280 B		1500 B	
		Mpps	Mbit/s	Mpps	Mbit/s	Mpps	Mbit/s	Mpps	Mbit/s
Baseline		20.83	13999.59	4.19	19965.90	1.92	19999.44	1.64	19927.58
Result #1		20.49	13770.09	4.19	19965.90	1.91	19861.92	1.63	19855.74
	rel to base	98.0 %		100.0 %		99.0 %		100.0 %	



Obrázek E.1: Příklad porovnání propustnosti dvou různých verzí knihovny DPDK u topologie P2P.

result_pvp_tput_pvp

Result ID	Frame size	64 B		576 B		1280 B		1500 B	
		Mpps	Mbit/s	Mpps	Mbit/s	Mpps	Mbit/s	Mpps	Mbit/s
Baseline		6.89	4627.96	3.27	15581.25	1.35	14076.49	1.13	13718.00
Result #1		6.68	4491.85	3.27	15581.25	1.35	14076.48	1.13	13718.01
	rel to base	97.0 %		100.0 %		100.0 %		100.0 %	



Obrázek E.2: Příklad porovnání propustnosti dvou různých verzí knihovny DPDK u topologie PVP.

Příloha F

Obsah CD

Obsah přiloženého CD má tuto strukturu:

```
/
├── codes/
│   ├── beaker_tasks/ ..... úlohy systému Beaker nutné pro spuštění testů
│   │   ├── build_moongen/ ..... úloha pro přípravu generátoru MoonGen
│   │   ├── perf_synchronization/ ..... implementace synchronizace
│   │   ├── setup_dpdk/ ..... skripty pro konfiguraci DPDK na straně generátoru
│   │   ├── sleeper/ ..... úloha pro rezervaci systému na zadanou dobu
│   │   └── vsperf_test/ ..... úloha automatizující spuštění nástroje VSperf
│   ├── cqe_api/ ..... knihovna pro komunikaci s aplikací CQE
│   ├── jenkins/ ..... konfigurační soubory pro systém Jenkins
│   ├── jobcreator/ ..... zdrojové soubory generátorů jobů pro systém Beaker
│   ├── lua-trafficgen/ .... lua skript implementující RFC2544 testování propustnosti
│   ├── reporting/ ..... skripty pro porovnávání výsledků testování
│   └── vsperf-project/ ..... zdrojové kódy oficiálního projektu VSperf
├── graphs/ ..... zdrojové soubory pro generování grafů do této práce
├── misc/ ..... ukázka HTML reportu a XML jobu pro systém Beaker
├── src/ ..... zdrojové soubory této diplomové práce
├── tests/ ..... ladicí soubory a výpisy z průběhu uskutečněných testů
├── README.txt ..... popis zprovoznění nástrojů a další informace
└── AUTORI.txt ..... informace autorství souborů v adresáři codes
```