VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZPRACOVÁNÍ PAKETŮ POMOCÍ KNIHOVNY DPDK

PACKET PROCESSING USING DPDK LIBRARY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ALEŠ PROCHÁZKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. MATĚJ GRÉGR, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání diplomové práce

Student: **Procházka Aleš, Bc.**
Program: Informační technologie Obor: Počítačové sítě a komunikace
Název: **Zpracování paketů pomocí knihovny DPDK**
Packet Processing Using DPDK Library
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se s knihovnou DPDK určenou pro práci s pakety.
2. Navrhněte aplikaci pro filtrování a přeposílání vybraných síťových paketů.
3. Implementuje navrženou aplikaci.
4. Změřte výkonnost dané aplikace, porovnejte s řešením standardního firewallu pomocí iptables.

Literatura:

- DPDK Docs, Data Plane Development Kit: Programmers Guide, September 2018, [online], url: http://dpdk.org/doc/guides/prog_guide/
- Bharadwaj, R. (2017). Mastering Linux Kernel development: A kernel developer's reference manual. ISBN: 978-1-78588-613-3.
- Robert Love. 2010. Linux Kernel Development (3rd ed.). Addison-Wesley Professional. ISBN: 978-0-672-32946-3

Při obhajobě semestrální části projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Grégr Matěj, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 22. května 2019
Datum schválení: 31. října 2018

Abstrakt

Tato diplomová práce se věnuje filtrování a přeposílání paketů ve vysokorychlostních sítích. Nejdříve je představen framework DPDK, který je využíván pro rychlé zpracování paketů. Dále je popsán návrh aplikace pro vysokorychlostní filtrování paketů a návrh nástrojů pro usnadnění práce s touto aplikací. Následně je představena implementace tohoto návrhu a v neposlední řadě testování a srovnání výsledků se standardním firewallem

Abstract

This master thesis focuses on filtering and forwarding packets in high speed networks. Firstly the DPDK framework is introduced, which is used for fast packet processing. This project also introduces a design of application for high-speed packet filtering and design of tools for making it easier to work with that application. Subsequently, the implementation of this design is introduced and testing with comparison of results with a standard firewall

Klíčová slova

DPDK, rychlé zpracování paketů, klasifikace paketů, firewall, filtrování provozu, ACL

Keywords

DPDK, fast packet processing, packet classification, firewall, traffic filtering, ACL

Citace

PROCHÁZKA, Aleš. *Zpracování paketů pomocí knihovny DPDK*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Matěj Grégr, Ph.D.

Zpracování paketů pomocí knihovny DPDK

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Matěje Grégra Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Aleš Procházka
22. května 2019

Poděkování

Tímto bych chtěl poděkovat mému vedoucímu práce panu Ing. Matěji Grégrovi, Ph.D. za jeho trpělivost, odbornou pomoc a cenné rady.

Obsah

Kapitola 1

Úvod

S přibývajícím počtem zařízení připojených do sítě Internet a neustálou zvyšující se propustností sítě nemusí běžné firewally postačovat ke kontrole provozu na síti. Například taková cloudová úložiště nebo streamovací služby produkují provoz několika desítek Gb/s. Chce-li pak například poskytovatel internetu kontolovat veškerý provoz procházející přes jeho hraniční routery, může u klasických firewallů dojít k zahlcení front, paměti, následnému zahazování paketů či dokonce k úplnému „zamrznutí“ a vyřazení služby z provozu. Z toho důvodu jsou potřeba aplikace, které umožní rychlé zpracování paketů. Tvorbě takové aplikace za použití frameworku DPDK, jež umožňuje rychlé zpracování paketů obejitím linuxového jádra, se věnuje tato práce.

Nejdříve je v kapitole 2 popsán framework DPDK a jeho principy, zejména jak je dosaženo rychlého zpracování.

Kapitola 3 popisuje zástupce standardního firewallu pracujícího na systémech Linux a Unix.

Kapitola 4 je věnována návrhu aplikace pro rychlé zpracování a přeposílání paketů, která bude využívat DPDK framework. V této kapitole jsou také navrženy některé nástroje pro následné usnadnění práce s touto aplikací, zejména práce se soubory, v nichž se nachází jednotlivá pravidla.

V kapitole 5 se pojednává o implementaci aplikace a jejích součástí navržených přechozí kapitolou. A kapitola 6 se věnuje testování a následnému porovnání výsledků se standardním firewallem iptables.

Kapitola 2

Popis DPDK

DPDK (Data Plane Development Kit) je sada knihoven a ovladačů síťových karet pro zrychlené zpracování paketů, běžící na široké škále procesorových architektur. Je navržena tak, aby běžela na procesorech x86, POWER a ARM. Běží převážně v uživatelské vrstvě Linuxu s FreeBSD portem pro podmnožinu DPDK funkcí. Je licencována pod open source BSD Licence. Tato sada knihoven je dostupná ke stažení na oficiálních stránkách[1].

DPDK implementuje model, kde všechny zdroje musí být alokovány před samotným voláním aplikací, které běží jako prováděcí jednotky na logických procesorových jádrech. Model nepodporuje plánovač a všechna zařízení jsou přístupná pomocí dotazování. Přerušení se nevyužívá kvůli režii výkonu způsobené přerušením zpracování [10].

Hlavní komponenty frameworku poskytují všechny prvky potřebné pro vysoce výkonné aplikace zpracovávající pakety. Mezi tyto komponenty patří např.[12]:

- **Ring manager (librte_ring)** – ring struktura poskytuje neomezené multi-producent, multi-konzument FIFO API v tabulce s konečnou velikostí. Je využívána memory pool managerem a může být použita jako hlavní komunikační mechanismus mezi jádry.
- **Memory pool manager (librte_mempool)** – je zodpovědný za alokaci souboru objektů v paměti. Tento soubor je identifikován jménem a využívá ring strukturu k uložení volných objektů. Poskytuje i několik volitelných služeb, např. vyrovnávací paměť pro objekty jednotlivých jader.
- **Network Packet Buffer Management (librte_mbuf)** – mbuf knihovna poskytuje možnost vytvořit a zrušit buffery, jež mohou být využívány DPDK aplikacemi k uložení bufferů pro zprávy. Buffery pro zprávy jsou vytvářeny při spuštění a uloženy v mempoolu (za pomoci mempool knihovny).
- **Timer Manager (librte_timer)** – tato knihovna poskytuje službu časovače k asynchronnímu vykonávání funkcí. Může se jednat o periodická nebo jednorázová volání. K získání přesného času využívá časového rozhraní vrstvy EAL a může být, v případě potřeby, inicializováno pro každé jádro zvlášť.

Vybrané komponenty budou blíže popsány v nadcházejících podkapitolách. Než budou rozebrány jednotlivé knihovny DPDK pro zpracování paketů, je vhodné se podívat, jakým způsobem probíhá zpracování paketu v Linuxovém jádře bez DPDK. Tomu se věnuje následující podkapitola 2.1.

[1] https://www.dpdk.org

2.1 Zpracování paketu v Linuxu

Jakmile síťová karta přijme paket, pošle ho do přijímací fronty a následně je provedena pomocí DMA (*Direct memory access* [2]) kopie do hlavní paměti. Poté musí být systém informován o novém paketu a předat data do speciálních bufferů (ty jsou alokovány pro každý paket). K tomuto linux využívá právě mechanismu přerušení - ta jsou při příchodu nového paketu do systému několikrát generována. Paket pak musí být přenesen do uživatelského prostoru.

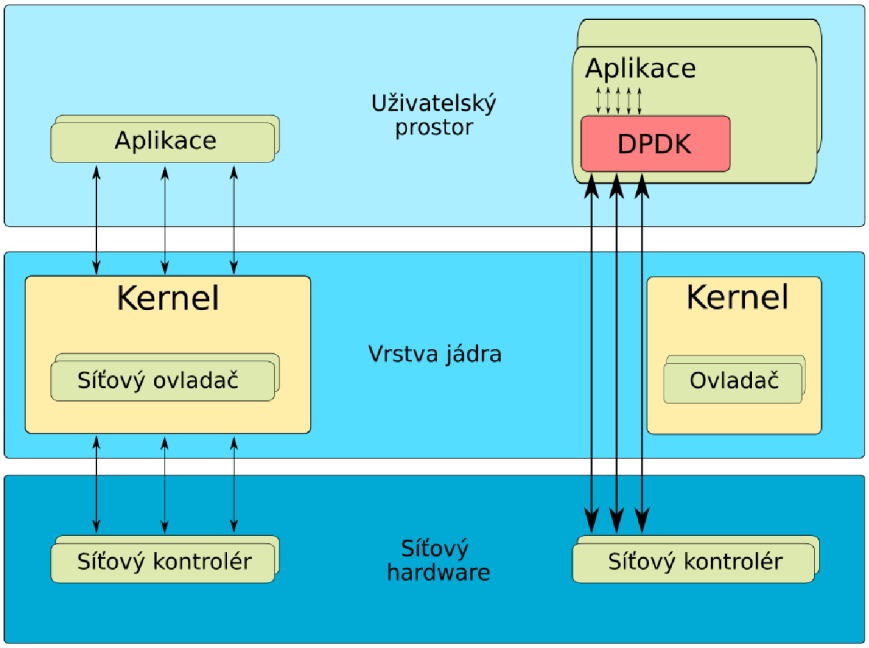
Z výše napsaného plyne jeden z problémů: čím více musí být zpracováno paketů, tím více se spotřebuje zdrojů. To negativně ovlivňuje výkonnost celého systému.

Jak již bylo řečeno, tyto pakety se ukládají do zvlášních struktur. Ty jsou vytvářeny pro každý paket a jsou uvolňovány až se paket přenese do uživatelského prostoru. Tato operace spotřebuje mnoho cyklů sběrnice - např. cykly přenášející data z procesoru do hlavní paměti.

Další negativní ovlivnění výkonu je přepínání kontextu. Když aplikace (v uživatelském prostoru) potřebuje poslat nebo přijmout paket, vykoná systémové volání. Kontext je přepnut do módu jádra a pak je vrácen do uživatelského módu - to spotřebuje značné množství systémových prostředků.

Některé z výše jmenovaných problémů se snaží řešit nové API (NAPI, v linuxových jádrech od verze 2.6), které kombinuje přístupy přerušení s dotazováním. Síťová karta nejprve pracuje v režimu přerušení, ale hned jak paket vstoupí na síťové rozhraní, zapíše se do fronty a zakáže přerušení. Systém pravidelné kontroluje fronty pro nová zařízení a shromažďuje pakety pro další zpracování. Jakmile je paket zpracován, karta bude odstraněna z fronty a přerušení je opět povoleno [20].

Pro lepší pochopení rozdílu zpracování paketu v Linuxu bez DPDK (levá část) a s DPDK (pravá část) slouží následující obrázek 2.1:



Obrázek 2.1: Zpracování paketu v Linuxu bez/s DPDK

[2] Metoda přímého přenosu dat mezi operační pamětí a vstupně/výstupními zařízeními.

2.2 Poll mode driver (PMD)

Běžné ovladače v linuxovém jádře využívají při příchodu paketu techniky obsluhy přerušení – jádro pozastaví svou činnost a zavolá obsluhu přerušení pro danou událost [2]. DPDK obsahuje své 1Gb, 10Gb, 40Gb a paravirtualizované ovladače. Tyto ovladače pracují v režimu dotazovaní (polling) – PMD přistupuje k deskriptorům vstupních a výstupních front přímo bez jakéhokoli přerušení. To umožní rychlé přijetí, zpracování a doručení paketu v uživatelské aplikaci [9].

PMD a síťová rozhraní

Každé síťové rozhraní je specificky označené svým PCI identifikátorem, ve formátu sběrnice:rozhraní.funkce. Tento identifikátor je přiřazen PCI enumerační funkcí při inicializaci DPDK. Rozhraním jsou na základě jejich PCI identifikátoru přiřazeny další dva identifikátory:

- **Index portu** používaný k označení rozhraní ve všech funkcích exportovaných pomocí PMD API
- **Název portu** používaný pro označení portu ve zprávách konzole, pro účely správy nebo ladění

Ethernetová zařízení mohou být vlastněny jednou DPDK entitou (např. aplikací, knihovnou, ovladačem atd.). Mechanismus vlastnictví je ovládán pomocí ethdev API a umožňuje nastavit, odstranit a získat vlastníka portu dle DPDK entity. To by mělo bránit vícenásobné správě ethernetového portu různými entitami [9].

PMD API

Ve výchozím stavu jsou všechny funkce exportované PMD funkce bez zámku, u nichž se nepředpokládá paralelní volání na různých logických jádrech k vykonání práce na stejném cílovém objektu. Například funkce přijímání nemůže být vyvolána na dvou logických jádrech pro dotazování stejné vstupní fronty na stejném portu. Tuto funkci lze samozřejmě volat paralelně různými jádry, ale pouze na různých vstupních frontách.

Paket je reprezentován strukturou `rte_mbuf`, což je obecná struktura metadat, obsahující všechny potřebné informace. To zahrnuje pole a stavové bity odpovídající hardwarovým funkcím, jako například výpočet kontrolního součtu IP hlavičky. Struktura `rte_mbuf` je blíže popsána v kapitole 2.8.

2.3 Vrstva EAL

EAL (Environment Abstraction Layer) je zodpovědná za získání přístupu k nízkoúrovňovým zdrojům, jako jsou např. hardware a paměťové zdroje. Poskytuje obecné rozhraní, které skrývá specifika prostředí aplikací a knihoven. Je odpovědná za inicializační rutiny a rozhodování, jak alokovat zdroje (paměťové zdroje, zařízení, časovače atd.). Typické služby očekáváné od EAL jsou např.[13]:

- **Načtení a spuštění DPDK:** DPDK a její aplikace jsou propojeny jako jedna aplikace a musí být načteny některými prostředky.

- **Afinita jádra:** EAL poskytuje mechanismus pro přiřazování jednotek výkomu určitým jádrům.
- **Rezervace systémové paměti:** EAL usnadňuje rezervaci různých paměťových zón, například oblasti fyzické paměti pro interakce zařízení.
- **Užitečné funkce:** spinloky a atomické čítače, které nejsou poskytovány v `libc`[3].
- **Trace a debug funkcí:** logy, dump_stack a další

DPDK aplikace jsou spouštěny v uživatelském prostoru pomocí knihovny `pthread`, každá prováděcí jednotka bude přiřazena určitému logickému jádru. Inicializace a spuštění jádra je prováděno ve funkci `rte_eal_init()` - to se skládá z volání do knihovny `pthread`. DPDK většinou přiřazuje jeden *pthread* na jádro. To umožňuje výrazné zvýšení výkonu, nicméně postrádá flexibilitu a není vždy efektivní.

Alokace velké souvislé fyzické paměti probíhá za pomoci kernelového souborového systému `hugetlbfs` funkcí `mmap()`. EAL poskytuje API k rezervaci pojmenovaných paměťových zón přímo v této souvislé paměti (fyzická adresa rezervované paměti je vrácena uživateli). Jsou dva režimy, ve kterých DPDK paměťový subsystém operuje:

- **Dynamický mód** - v tomto režimu bude využití velkých paměťových stránek (hugepages) DPDK aplikací růst a zmenšovat na základě požadavků aplikace.
- **Legacy mód** - tento mód napodobuje historické chování EAL. To znamená, že je rezervována všechna paměť při startu a není umožněno získání či uvolnění velkých paměťových stránek (hugepages) za běhu ze systému.

Sdílené proměnné jsou výchozí chování. Lokální proměnné pro jednotlivá vlákna jsou implementována díky Thread Local Storage (TLS) - poskytuje lokální úložitě pro každé vlákno.

Vrstva EAL také poskytuje *malloc API* k alokaci paměti jakékoliv velikosti. Cílem tohoto API je poskytnou funkce (podobné `malloc`), které umožňují alokaci paměti z prostoru velkých paměťových stránek.

2.4 Velké paměťové stránky (hugepages)

Vždy, když proces pracuje s nějakou pamětí, musí procesor označit RAM jako využívanou právě tímto procesem. Z důvodů efektivity procesor alokuje RAM po úsecích velikosti 4KB - tyto úseky se nazývají stránky (*pages*). Jelikož je adresový prostor virtuální, musí si procesor a operační systém pamatovat, které stránky patří jakým procesům a kde jsou jsou tyto stránky uloženy. Je tedy zřejmé, že čím více stránek je naalokovaných, tím více zabere času, než je nalezeno místo, kde je tato paměť mapována - to může citelně ovlivnit výkonnost aplikace. Většina nynějších procesorových architektur podporuje větší stránky, jež se v Linuxu nazývají *huge pages* (*super pages* v BSD a *large pages* ve Windows). Větší velikosti stránek mají za následek méně alokovaných bloků, tudíž rychlejší vyhledávání, což může pozitivně ovlivnit výkonnost aplikace. [15, 17].

DPDK vyžaduje podporu velkých paměťových stránek pro alokaci velkých datových struktur k ukládání paketových bufferů. Nejpoužívánější velikosti jsou 2MB nebo 1GB (pro 64 bitové aplikace se doporučuje využít velikost 1GB, pokud je podporováno platformou)[3].

[3]Standardní knihovna jazyka C

2.5 Spouštěcí parametry

Při spouštění DPDK aplikací se využívají dvě části parametrů. V první části se předávají parametry knihovně EAL a druhá část patří aplikačním parametrům (jako oddělovač slouží dvě pomlčky). Spouštění aplikace vypadá obecně následovně:

```
./DPDK-app <parametry pro EAL> -- <parametry pro aplikaci>
```

Parametry pro aplikaci jsou specifické pro každou aplikaci, kdežto prametry pro knihovnu EAL jsou u každé DPDK aplikace stejné. Následuje výčet některých důležitých parametrů pro nastavování EAL [6, 7]:

-c *COREMASK* nebo **-l *CORELIST***: Hexadecimální bitová maska jader, která budou využita pro běh aplikace. `CORELIST` je seznam čísel jader namísto bitové masky.

-n *<number of channels>*: počet paměťových kanálů na procesorový socket.

-b *<domain:bus:devid.func>*: zamezuje EALu využít specifikované PCIe zařízení. Je možné využít více těchto přepínačů.

--use-device: použití pouze zadaných ethernetových zařízení. Nelze využít s přepínačem `-b`

--socket-mem *<amounts of memory per socket>*: paměť k alokování na daných socketech z velkých paměťových stránek.

--socket-limit *<amounts of memory per socket>*: vymezení maximální dostupné paměti k alokaci na každém socketu.

-d: přidání ovladače nebo adresáře ovladače k načtení. Tento parametr se využívá k nahrání PMD ovladačů, které jsou sestaveny jako sdílené knihovny.

-m *<megabytes>*: paměť, kterou lze přiřadit z velkých paměťových stránek bez ohledu na proserovoé sockety. Místo tohoto přepínače je doporučeno užívat `--socket-mem`.

-r *<number of ranks>*: nastavuje počet paměťových řad (ve výchozím nastavení je automaticky detekováno).

-v: zobrazí informace o verzi DPDK.

--huge-dir *<path to hugetlbfs directory>*: určení adresáře, kde jsou připojené velké paměťové stránky.

--file-prefix *<prefix name>*: použití různých prefixů pro názvy souborů velkých paměťových stránek. Tento přepínač umožňuje běh několika nezávislých DPDK procesů.

--proc-type *<primary|secondary|auto>* : nastavení typu aktuálního procesu.

--base-virtaddr *<address>*: pokus o použití jiné počáteční adresy pro všechny paměťové mapy primárního DPDK procesu. Může to být užitečné, pokud nemůže být spuštěn druhý proces kvůli konfliktům v adresové mapě.

--legacy-mem: spustí DPDK v paměťovém režimu legacy, zakáže tak rezervování a uvolňování paměti za běhu programu.

--single-file-segments: vytvoří méně souborů ve velkých paměťových stránkách (pouze v dynamickém režimu).

Povinné jsou pouze přepínač `-c` (nebo `-l`), všechny ostatní jsou volitelné.

2.6 Ring knihovna

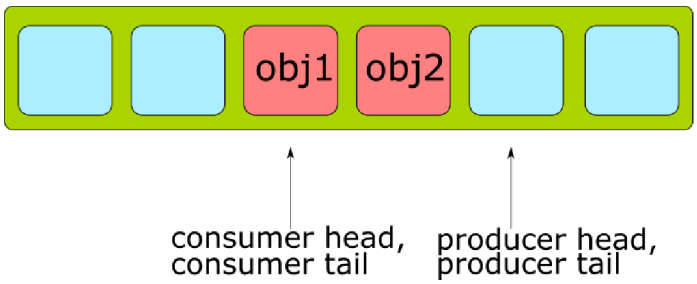
Tato knihovna umožňuje správu front. Namísto používání nekonečného seznamu má knihovna následující vlastnosti [5]:

- FIFO fronta
- Maximální velikost je pevná (v tabulce jsou uchováváni ukazatelé)
- Implementace bez zámků
- Multi-konzument nebo single-konzument odebrání z fronty
- Multi-producent nebo single-producent přidání do fronty
- Bulk enqueue/dequeue - přidání/odebráni z fronty zadaného množství objektů
- Burst enqueue/dequeue - přidání/odebrání z fronty maximálního počtu dostupných objektů

Hlavní výhody oproti vázanému seznamu tvoří **rychlost** (pouze jedna instrukce na porovnání a přehození), **jednoduchost**, **přizpůsobení hromadným operacím** přidání/odebrání z fronty.

Mezi nevýhody patří pevná velikost, držení několika struktur typu ring stojí více paměti než vázaný seznam (prázdný ring obsahuje alespoň N ukazatelů).

Zjednodušená reprezentace této struktury (včetně ukazatelů) je zobrazena na obrázku 2.2.



Obrázek 2.2: Ring struktura

2.7 Mempool knihovna

Memory pool je alokátor objektů pevné velikosti, identifikován je jménem a využívá *mempool handler* k uložení volných objektů (výchozí *handler* je založený na struktuře ring). Poskytuje též volitelné služby jako je mezipaměť pro objekty jednotlivých jader nebo zarovnávání objektů pro rovnoměrné šíření skrz DRAM nebo DDR3 kanály. Tato knihovna je užívána knihovnou `Mbuf`, jejíž popis je v následující sekci 2.8.

2.8 Mbuf knihovna

Tato knihovna poskytuje možnost alokovat a uvolňovat buffery, jež bývají používány k uchovávání bufferů zpráv. Tyto buffery jsou uloženy v *mempool*. Struktura `rte_mbuf` je obecně využívána jako buffer síťových paketů, nicméně ve skutečnosti může nést jakákoliv data (například řídicí data, události atd.). Struktura hlavičky je udržována co nejmenší a v současné době používá pouze dva řádky vyrovnávací paměti. Všechny síťové aplikace by měly používat *mbuf* pro přenos síťových paketů [8].

Tato knihovna umožňuje manipulaci s daty v *mbuf* paketu za pomoci některých funkcí. Např:

- získat ukazatel na začátek dat
- získat délku dat
- přidat před původní data nová data
- přidat nová data za původní data
- odstranit data ze začátku/konce bufferu

Návrh paketových bufferů

Pro ukládání dat z paketu (včetně hlaviček protokolů) byly zvažovány dva přístupy [8]:

1) Udržovat metadata v rámci jediného bufferu následovaného oblastí pevné velikosti pro data paketu
2) Použít oddělené paměťové buffery pro strukturu metadat a pro data paketu.

Výhoda prvního přístupu tkví v potřebě pouze jedné operace k alokaci/uvolnění celé paměti reprezentující paket. Kdežto druhá metoda je více flexibilní a povoluje celkové oddělení alokace struktur metadat od alokace bufferů pro data paketu.

Pro DPDK byla vybrána první možnost. Metada obsahují řídicí informace, jako jsou typ zprávy, offset k začátku dat a ukazatel pro dodatečné struktury *mbuf* umožňující zřetězení bufferů.

Přímé a nepřímé buffery

Přímý buffer je kompletně oddělený a samostatný. **Nepřímý buffer** se chová jako přímý, ale ve skutečnosti ukazatel na buffer a odsazení dat v něm odkazují na jiný přímý buffer. Toho se využívá v situacích, kdy je potřeba duplikovat či fragmentovat pakety.

Buffer se stává nepřímým, když je připojen k přímému voláním funkce `rte_pktmbuf_attach()`. Každý buffer má počítadlo referencí a pokaždé, když je připojen k přímému bufferu se počítadlo inkrementuje. Obdobně se při odpojení dekrementuje. Pokud je počítadlo rovno 0, přímý buffer je uvolněn, jelikož již není využíván [8].

2.9 ACL knihovna - klasifikace paketů a řízení přístupu

DPDK poskytuje knihovnu pro řízení přístupu, která umožňuje klasifikovat vstupní paket na základě klasifikačních pravidel.

ACL knihovna se využívá k vyhledávání n-tic nad sadou pravidel s více kategoriemi a pro každou kategorii najde nejlepší shodu. Knihovní API poskytuje následující základní operace [4]:

- vytvořit nový kontext
- přidat pravidla do kontextu
- provést klasifikaci vstupního paketu
- pro všechna pravidla v kontextu vytvořit struktury potřebné ke klasifikaci
- zrušení kontextu a jeho struktur a uvolnění přiřazené paměti

Definice pravidel

Současná implementace umožňuje uživateli pro každý kontext určit vlastní pravidlo. Nicméně je zde několik omezení, převážně z důvodu výkonu [4]:

- první pole v pravidle musí být jeden byte dlouhé
- všechna následující pole musí být seskupeny do sad čtyř po sobě jdoucích bajtů

K definici každého pole uvnitř pravidla se využívá struktura obsahující:

- ***Type*** - pole je jedno ze tří mmožností:
  - *_MASK* - pro pole jako IP adresy, jež mají hodnotu a masku definující počet relevantních bitů
  - *_RANGE* - pro pole jako jsou porty, které mají nížší a vyšší hodnotu
  - *_BITMASK* - pro pole identifikující protokol, které mají hodnotu a bitovou masku
- ***Size*** - definuje délku pole v bajtech
- ***Field_index*** - hodnota reprezentující pozici pole v pravidle, nabývá hodnot 0 až N-1 pro N polí
- ***Input_index*** - specifikuje, do které vstupní skupiny pole patří
- ***Offset*** - definuje offset pro pole - offset od začátku bufferu pro hledání

Při vytváření sady pravidel je nutné pro každé pravidlo přidat další informace

- **prioritu** - váha pro měření priority pravidel (vyšší je lepší). Pokud dojde ke shodě s více pravidly, je vráceno pravidlo s větší prioritou.
- **masku kategorie** - každé pravidlo používá bitovou masku pro výběr příslušné kategorie daného pravidla. Toto umožňuje efektivní paralelní vyhledávání.
- **uživatelská data** - uživatelsky definovaná hodnota, která je vrácena při shodě. Pokud ke shodě nedojde, je vrácena nula

Velikostní limit paměti

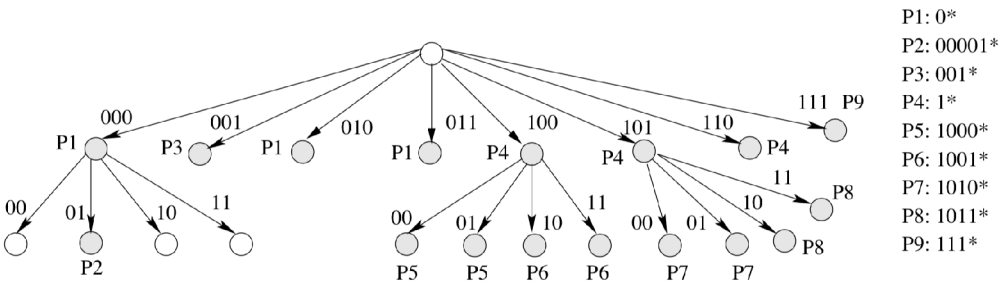
Build fáze vytváří pro danou sadu pravidel vnitřní strukturu pro další běh. V současné implementaci se jedná o sadu tzv. *„multibit trie“* (s krokem 8), popis struktury *„multibit trie“* bude následovat v dalším odstavci. Dle nastavených pravidel, jež by mohly spotřebovávat značné množství paměti. Při pokusu o úsporu paměti se sestavovací proces pokusí danou sadu rozdělit do několika neprotínajících se podmnožin a vytvořit pro každou z nich samostatnou trie. V závislosti na sadě pravidel to může snížit požadavky na paměť, ale zase to také může zvýšit dobu klasifikace. V době sestavování je možné určit maximalní limit paměti pro vnitřní struktury daného kontextu. Nastavením hodnoty větší než nula říkáme sestavovacímu procesu:

- Pokus se o minimalizaci počtu trie v tabulce, ale
- Ujisti se, že velikost tabulky by nepřekročila zadanou hodnotu

Nastavením hodnoty nula, použije sestavovací proces výchozí chování - zkusí minimalizovat velikost struktur, ale nevystavuje na něm žádný pevný limit. To dává uživateli možnost rozhodovat o poměru výkonu a prostoru. [4]

Multibit trie

Trie je binární stromová struktura, používaná pro ukládání prefixů (odtud také název *„prefixový strom“*). Uzly tvoří části uložené informace (například IP adresa), ohodnocené hrany odpovídají jednotlivým bitům informace. Výška stromu odpovídá nejdelšímu prefixu. Vícebitové trie pak umožňují porovnávat více bitů v jednom kroku. Při porovnávání čtyř bitů najednou se počet přístupů do paměti zredukuje z 32 na 8 (např. při vyhledávání jedné IPv4 adresy). Počet bitů, které se kontrolují během jednoho porovnání, se nazývá krok (*„stride“*)[16]. Příklad vícebitového stromu trie je graficky znázorněn na obrázku 2.3.A jak bylo uvedeno v předchozím odstavci - DPDK využívá vícebitové trie s krokem 8.



Obrázek 2.3: Vícebitový strom trie s pevnou délkou kroku. Převzato z [16]

Kapitola 3

IPtables

Na samém konci této diplomové práce bude prováděno měření výkonnosti implementované aplikace s využitím DPDK frameworku a následné porovnávání se standardním firewallem `iptables`. V této kapitole bude stručně představen nástroj `iptables`.

Subsystém pro zpracování paketů v linuxovém jádře se nazývá `netfilter` a `iptables` je konzolový program v uživatelském prostoru používaný k jeho konfiguraci. Architektura `iptables` seskupuje pravidla pro zpracování paketů do tabulek podle funkcí (filtrování paketů, překlad síťových adres a jiné změny paketů), z nichž každá má řetězce pravidel pro zpracování. Pravidla se skládájí ze shod (*matches* – určení, pro které pakety platí) a cílů (*targets* – určují, co se s odpovídajícím paketem stane) [18].

3.1 Koncept

`iptables` definuje celkem pět záchytných bodů v cestách při zpracovávání paketů [18]:

- PREROUTING – umožňuje zpracovávat pakety tak, jak přicházejí ze síťového rozhraní
- INPUT – zpracování paketů těsně před doručením lokálnímu procesu
- FORWARD – umožňuje zpracovávat pakety, které jdou přes bránu, přicházející jedním rozhraním a jdoucí zpět do druhého
- POSTROUTING – zpracování paketů před jejich opuštěním síťového rozhraní
- OUTPUT – zpracování paketů po vygenerování lokálním procesem

Vestavěné řetězce jsou k těmto bodům připojeny – je možné přidat sekvenci pravidel pro každý záchytný bod, kde každé pravidlo představuje příležitost ovlivnit či sledovat tok paketu.

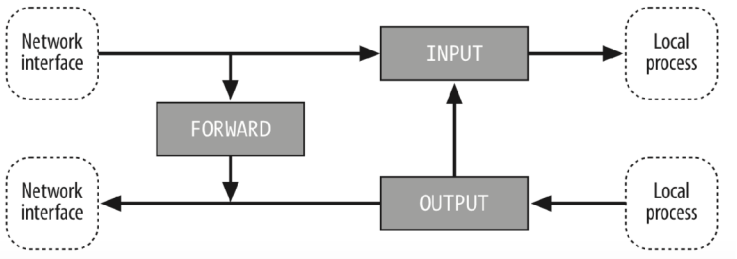
3.2 Typy tabulek

Existují tři typy tabulek, se kterými `iptables` pracuje [18]:

- filter – používané pro nastavení pravidel pro směry povolené komunikace do, skrz a ven z počítače. Vestavěné záchytné body jsou: FORWARD, INPUT a OUTPUT

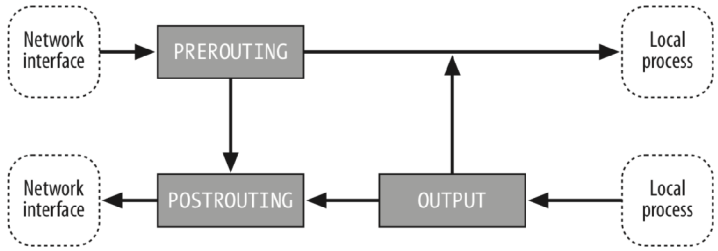
- mangle – používá se pro specializované paketové změny, jako např. rozdělení *IP options* z hlavičky. Vestavěné záchytné body jsou: FORWARD, INPUT, OUTPUT, POSTROUTING a PREROUTING
- nat – slouží pro překlad adres. Vestavěné záchytné body jsou: OUTPUT, POSTROUTING a PREROUTING

Na následující sekvenci obrázků jsou popsané záchytné body znázorněny v jednotlivých tabulkách. Obrázek 3.1 zobrazuje průchod paketu systémem při filtrování paketů (tabulka *filter*).



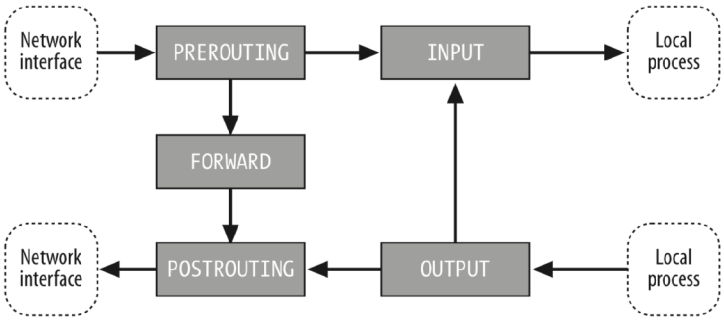
Obrázek 3.1: Tok paketu při filtrování. Převzato z [18]

Obrázek 3.2 ukazuje průchod paketu systémem při překladu adres (tabulka *NAT*).



Obrázek 3.2: Tok paketu při překladu adres NAT. Převzato z [18]

A obrázek 3.3 naznačuje průchod paketu systémem pro změny definované v tabulce *mangle*.



Obrázek 3.3: Tok paketu pro tabulku *mangle*. Převzato z [18]

3.3 Pravidla

Pravidlo v `iptables` se skládá z jednoho či více výběrového kritéria určující pakety, které jím budou ovlivněny (musí být splněny všechny možnosti shody, aby paket odpovídal pravidlu) a z upřesnění, jak bude s daným paketem naloženo.

Systém udržuje počítadla pro každé pravidlo. Vždy, když paket splňuje kritéria pravidla, inkrementuje se počítadlo paketů a počítadlo bajtů je zvětšeno o velikost shodujícího se paketu. Pokud pravidlo neobsahuje informaci o tom, jak má být s odpovídajícím paketem naloženo, je paket zpracován obvyklým způsobem, jakoby pravidlo neexistovalo (jen jsou aktualizována počítadla)[18].

Pro specifikování akce s paketem při splnění pravidla se využívají vestavěné politiky:

- ACCEPT - paket je přijat a poslán do další fáze zpracovávání
- DROP - úplné přerušení zpracovávání paketu.
- QUEUE - pošle paket do uživatelského prostoru
- RETURN - z uživatelsky definovaného pravidla ukončí zpracovávání a pokračuje dalším pravidlem v pčedchozím řetězci

Jak již bylo řečeno, výchozí akce pro paket je v `iptables` politika ACCEPT. Některé lepší, ale složitější, firewally mění tuto politiku na DROP. Tímto můžeme například povolit pouze jednu službu, typicky port 80 pro webový server. Výchozí DROP politiky je využíváno například v Cisco ACL.

V úvodu této kapitoly bylo řečeno, že `iptables` je konzolový nástroj. To znamená, že pravidla pro tabulky jsou přidávána pomocí příkazů v příkazové řádce. Příkaz pro přidání pravidla může vypadat následovně:

```
iptables -t filter -A INPUT -s 192.168.1.1 -p icmp -j DROP
```

Toto pravidlo říká, že do tabulky *filter* je přidán záchytný bod INPUT pro paket se zdrojovou adresou 192.168.1.1 a protokolem ICMP a takový paket bude zahozen.

Příkazy pro odstranění pravidla vypadají obdobně:

```
iptables -t filter -D INPUT 1

iptables -t filter -D INPUT -s 192.168.1.1 -p icmp -j DROP
```

Prvním pravidlem se maže první pravidlo v tabulce *filter* se záchytným bodem INPUT. Druhé pravidlo přesně specifikuje pravidlo, které má být smazáno. Význam jednotlivých parametrů je stejný jako při přidávání pravidla, pouze při přidávání se používá `-A` (jako *add*) a při mazání `-D` (jako *delete*).

Pro přehled všech použitelných parametrů je vhodné navštívit manuálové stránky[1].

[1] V příkazové řádce zadat `man iptables`

Kapitola 4

Návrh aplikace

V této kapitole bude představen návrh aplikace pro filtrování a přeposílání vybraných síťových paketů s využitím knihoven frameworku DPDK.

4.1 Konfigurační soubor

Jelikož se při spouštění aplikace může využívat velkého množství parametrů, které jsou navíc rozdělené na dvě části, kde první část jsou parametry pro EAL a druhá část jsou parametry pro aplikaci samotnou, jsou jednotlivé přepínače pro zjednodušení spouštění aplikace zapisovány do konfiguračního souboru. Soubor se nachází ve stejném adresáři jako spouštěcí skript (viz dále) a aplikace samotná. V konfiguračním souboru se mimo jiné bude zadávat cesta k souborům s pravidly pro vyhodnocení paketů. Tyto soubory budou dva - pro IPv4 a IPv6 zvlášť.

4.2 Pomocné skripty

S aplikací pro filtrování a přeposílání paketů jsou současně implementovány skripty usnadňující práci s výsledným programem a hlavně také s pravidly v souborech.

Spouštěcí skript

Spouštěcí skript bude sloužit ke snadnějšímu spouštění přeložené aplikace. Hlavním smyslel tohoto skriptu by mělo být, že sám automaticky provede změny ovladačů, které jsou podporovány frameworkem DPDK na síťových rozhraních, jež budou zadány v konfiguračním souboru a tedy využívány hlavní aplikací. Bude používat zapsané parametry v konfiguračním souboru, se kterými bude spuštěn přeložený program.

Pomocné nástroje addRule4 a addRule6

Jedná se o dva jednoduché skripty, jimiž lze přidávat jednotlivá pravidla pro filtrování a přeposílání. Skript `addRule4` přidává IPv4 pravidla a `addRule6` IPv6 pravidla. Hlavním smyslem těchto skriptů je kontrola, aby pravidlo bylo zapsáno ve správném formátu, jelikož při manuálním vkládání přímo do souboru může snadno dojít k chybě. Skripty nabízejí nápovědu, která zobrazuje jaký očekává formát pravidla. Úspěšně či neúspěšné vložení pravidla je oznámeno hláškou. Po jakékoli manipulaci pravidel v souborech (ať už manuální

nebo pomocí utilit) je nutné celou DPDK aplikaci restartovat, aby došlo k načtení nových/editovaných pravidel do ACL kontextu. Cesta k souborům bude předávána argumentem při spouštění skriptu.

Skript checkRules

Tento skript zkontroluje, zda jsou pravidla v souborech správně zapsány. Je doporučeno tento skript použít po editaci souboru s pravidly. Cesta k souborům s pravidly bude předávána argumentem při spouštění skriptu.

4.3 Definice pravidel

Definice pravidel je zapsána ve dvou textových souborech (jeden pro IPv4 adresy a druhý pro IPv6 adresy). Tyto soubory jsou ve výchozím stavu ve složce `/ruleset`, která se nachází ve stejném adresáři jako aplikace (cesty k těmto souborům jsou předdefinovány v konfiguračním souboru). Do těchto souborů lze připisovat pravidla manuálně nebo je možné využít přiložených nástrojů `addRule4` a `addRule6`.

Aplikace využívá dvou druhů pravidel:

- ACL pravidla – jsou využívána pro **filtrování/zahazovaní** paketů
- Směrovací pravidla – jsou využívána pro **přeposílání/směrování** paketů

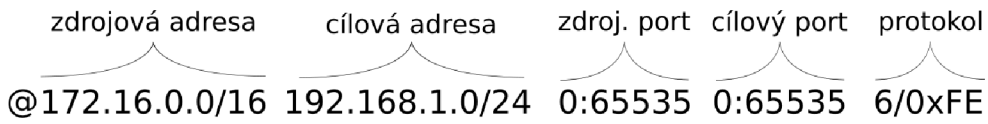
ACL pravidla jsou volitelná, zatímco směrovací pravidla jsou povinná (alespoň jedno pravidlo této kategorie by mělo být přítomno v každém souboru). Aby bylo pravidlo aplikací správně parsováno, musí splňovat následující podmínky:

- každé pravidlo na jednom řádku
- jsou možné pouze následující typy řádků
  - ACL pravidla – začínající znakem `@` (zavináč)
  - směrovací pravidla – začínající znakem `R`
  - komentář – označovaný znakem `#`
  - prázdný řádek

Všechny ostatní typy řádků jsou označeny jako nevalidní a aplikace skončí s chybou. Pro kontrolu správnosti zapsaných pravidel bude sloužit přiložený nástroj `checkRules`. O nesprávném či správném formátu souboru je uživatel informován hláškou na obrazovce.

Formát pravidel

Typické filtrovací pravidlo je zobrazeno na následujícím obrázku 4.1:

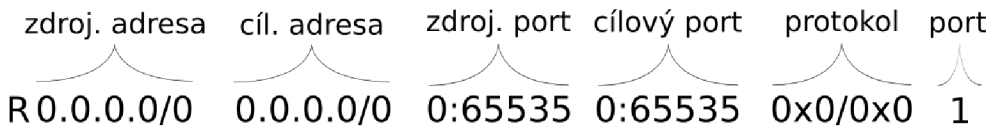


Obrázek 4.1: Příklad ACL pravidla

Pravidlo z obrázku 4.1 nám říká, že se jedná o pravidlo, které bude pakety zahazovat (určuje znak `@`). Zahazovat se budou ty pakety které splňující následující:

- zdrojová adresa je v rozsahu 172.16.0.0 – 172.16.255.255
- cílová adresa je v rozsahu 192.168.1.0 – 192.168.1.255
- zdrojový port je z rozsahu 0 – 65535
- cílový port je z rozsahu 0 – 65535
- protokol nabývá hodnot 6 nebo 7 (výpočet bude následovat dále)

Typické pravidlo pro směrování pro IPv4 ukazuje následující obrázek 4.2:



Obrázek 4.2: Příklad směrovacího pravidla

Toto pravidlo vypadá obdobně jako předchozí. Pouze je tu znak `R`, který označuje směrovací pravidla a zadává se identifikátor portu, na který se bude provoz splňující pravidlo přesměrovávat. Zápis z obrázku 4.2 říká, že všechny pakety se budou přesměrovávat na port 1.

Jak vypadá komentář je ukázáno na obrázku 4.3.

```
#Tohle je komentář
```

Obrázek 4.3: Příklad komentáře

Výpočet hodnoty protokolu

Identifikátor protokolu (`hodnota/maska`) je osmibitové pole skládající se z hodnoty a masky, které pokrývají rozsah hodnot. Pro kontrolu, zda hodnota náleží do rozsahu, slouží následující výraz:

$$(zvolena\_hodnota \ \& \ maska) == hodnota \tag{4.1}$$

Řekněme, že máme zadaný protokol výrazem 6/0xfe, tak jak bylo definováno na obrázku 4.1, a chceme zjistit, zda do rozsahu patří čísla protokolů 7 a 8. Kontrolní výpočet by tedy byl následující:

zvolená hodnota = 7	zvolená hodnota = 8
maska = 0xfe	maska = 0xfe
hodnota = 6	hodnota = 6
0000 0111	0000 1000
& 1111 1110	& 1111 1110
0000 0110	0000 1000
Výsledek: 6	Výsledek: 8

Jak je vidět z výpočtu, tak číslo 7 patří do rozsahu, nicméně číslo 8 již ne, neboť nám po operaci *and* nevyšla původní hodnota (v tomto případě číslo 6).

Způsob výpočtu identifikátoru protokolu (hodnoty a masky) je stejný jako například u počítání IP adres z hodnoty IP adresy sítě a masky. Rozsah může být vyčíslen v binárních číslech s bity, které se nikdy nezmění a bity dynamicky se měnícími. Ty bity, které se dynamicky změnily jsou nastaveny v masce a hodnotě na 0. Nikdy nezměněné bity jsou v masce nastaveny hodnotou 1 a v hodnotě očekávaným číslem. Například rozah 6 až 7 je vyčíslen jako 00000110 (číslo 6) a 00000111 (číslo 7) binárně. Bity 1 až 7 jsou bity, které se neměnily a bit 0 je bit dynamicky se měnící. Proto je v masce a hodnotě bit 0 nastaven hodnou 0 a bity 1 až 7 v masce 1. Bity 1 až 7 v hodnotě jsou nastaveny číslem 0000011. Z toho vyplývá, že maska je 0xFE hexadecimálně (to se rovná 11111110 binárně) a hodnota je 0x6 hexadecimálně (00000110 binárně).

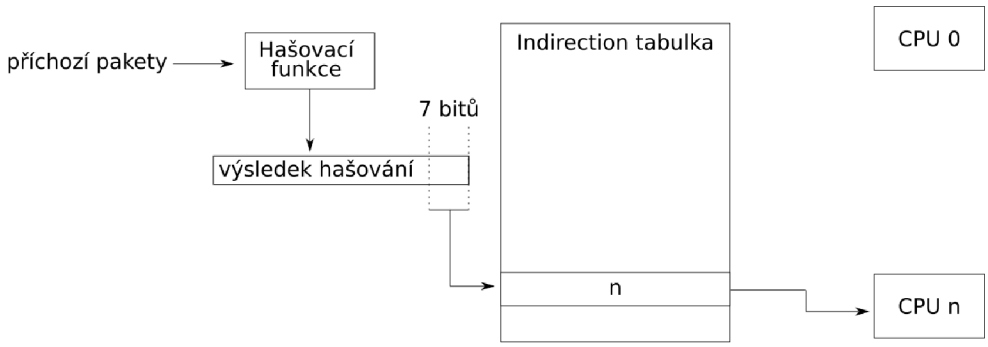
Priorita pravidel

Priorita pravidla je určena jeho pozicí v souboru s tím, že jsou pravidla uspořádána v sestupném pořadí priorit, tzn. že pravidlo na začátku souboru má vždy vyšší prioritu než pravidla uvedena níže v souboru. Kdyby se vstupní paket shodoval s více pravidly, je vždy vráceno to s větší prioritou.

4.4 RSS (Receive side scaling)

Díky této technoligii jsme schopní zpracovávat pakety na více jádrech procesoru. Ve výsledné aplikaci budou jednotlivá jádra vykonávat stejnou práci, tzn. všechna budou zpracovávat pakety, porovnávat je s pravidly a rozhodovat, zda je zahodit či poslat dál. Rozdíl spočívá v tom, s jakými pakety budou konkrétní jádra pracovat.

Rozdělení paketů do front jednotlivých jader zaručí technologie RSS. To aplikuje hašovací funkci na každý příchozí paket, z kterého použije informace z hlaviček síťové a transportní vrstvy, např. IP adresy a TCP porty paketu. Z této vypočtené hodnoty je vymaskováno spodních sedm bitů, které udávají index do *indirection* tabulky, kde jsou uložena čísla front procesorů (hodnota je použita k přiřazení paketu procesoru. [14, 19]. Následující diagram 4.4 znázorňuje průběh RSS.



Obrázek 4.4: Diagram výpočtu RSS

4.5 Zpracování a vyhodnocení paketů

Aplikace podporuje IPv4 i IPv6 adresy. Aby bylo možné provádět vyhodnocování a následné přeposlání či zahození paketů, je nutné extrahovat z paketu **pětici**, se kterou se bude dále pracovat. Tato pětice obsahuje:

- zdrojovou a cílovou IP adresu
- zdrojový a cílový port
- protokol

Databáze pravidel ACL knihovny

Po přijetí paketu jádrem se vyhledává v databázi uložených pravidel. Tato databáze je reprezentována pomocí takzvané *„multibit trie“* (jak již bylo popsáno v kapitole 2.9). Tyto trie jsou vytvořeny dvě - pro IPv4 a IPv6 pravidla zvlášť. Pokud se paket shoduje s ACL pravidlem (viz. 4.3), je tento paket zahozen. Shoduje-li se se směrovacím pravidlem (viz. 4.3) je tento paket přeposlán na daný port. Pokud se paket neshoduje s žádným pravidlem, je zahozen.

Kapitola 5

Implementace aplikace

Tato kapitola popisuje implementaci aplikace, jejíž návrh byl popsán v kapitole 4. Program je napsán v programovacím jazyce C (v tomtéž jazyce je napsán i framework DPDK). Podpůrné miniprogramy a spouštěcí skript jsou napsány v jazyce Python, popis jejich implementace následuje v podkapitole 5.2. Výsledná aplikace je implementována a testována s frameworkem DPDK ve verzi 18.05 a vychází z několika příkladů užití DPDK knihovny, dostupných přímo v instalačním adresáři pod licencí BSD.

5.1 Vstupní parametry programu

Aplikace se spouští s parametry pro nastavení vrstvy EAL a aplikačními parametry. Nejpoužívanější argumenty vrstvy EAL jsou popsány v kapitole 2.5. Pro běh aplikace postačují parametry nastavující počet jader (parametr `-c` pro masku jader nebo `-l` pro seznam jader) a počet paměťových kanálů (paramater `-n`). Aplikační parametry jsou tři:

- `-p` *`PORTMASKA`* - slouží k nastavení hexadecimální masky interfaců, jež budou použity v aplikaci. Například maska 0x3 říká, že budou využity porty s indexem 0 a 1.
- `--ipv4-rule-path` *`SOUBOR`* - cesta k souboru s pravidly pro IPv4
- `--ipv6-rule-path` *`SOUBOR`* - cesta k souboru s pravidly IPv6

Před samotným spuštěním aplikace je nutné mít požadovaná síťová rozhraní svázaná s ovladači podporovaných knihovnou DPDK. Proto je vhodné zapisovat parametry do konfiguračního souboru a aplikaci spouštět pomocným skriptem, který je schopen zajistit toto svázání. Popis skriptu bude následovat.

5.2 Implementace utilit

Jak již bylo avizováno dříve, jsou s výslednou aplikací vyvinuty i další podpůrné skripty, které usnadňují práci s pravidly nebo spouštění aplikace. Spouštěcí skript využívá konfigurační soubor, jehož popis nyní následuje.

Konfiguračná soubor

V konfiguračním souboru se uvádí parametry pro nastavení vrstvy EAL, názvy síťových rozhraní, název ovladače, se kterým budou rozhraní navázána a cesty k souborům s pravidly

pro filtrování a přeposílání paketů. Tento soubor **nevyužívá** aplikace samotná, nýbrž pouze spouštěcí skript. Soubor nese název `config.ini` a nachází se ve stejné složce jako aplikace a spouštěcí skript.

Jednotlivé parametry jsou rozděleny do dvou sekcí:

- **[EAL]** - parametry k nastavení vrstvy EAL. Zápis ve formátu: parametr = hodnota. Názvy parametrů se využívají stejné, jako jsou uvedeny např. v kapitole 2.5
- **[application]** - v této sekci se nastavují cesty k souborům s pravidly, seznam názvů síťových rozhraní, které má aplikace využívat a název ovladače podporovaného frameworkem DPDK, s nímž mají být síťová rozhraní svázána. Ve výchozím stavu je nastaven ovladač `igb_uio`[1].

Pro příklad, jak takový soubor může vypada slouží následující obrázek.

```
[EAL]
;memory channels
n = 4
;lcore list
l = 0

[application]
interfaces = enp0s5 enp0s6
driver = igb_uio
ipv4-rules = ./ruleset/ipv4_rules.db
ipv6-rules = ./ruleset/ipv6_rules.db
```

Obrázek 5.1: Ukázka jednoduchého konfiguračního souboru

Utilita checkRules

Tento skript je napsán v jazyce Python a otestován s verzí interpreteru 3.6. Tato utilita se nachází v podadresáři `tools` a soubor se jmenuje `checkRules.py`.

Skript prochází soubory s pravidly, jejichž cesta je mu zadána pomocí parametrů při spuštění, řádek po řádku a kontroluje, zda jsou jednotlivé řádky zapsány ve správném formátu. V případě nalezení chyby si ukládá informace do struktury typu slovník, kde klíčem je číslo řádku a hodnota je chybová hláška. Tento slovník je vytvořen pro každý soubor. Po zkontrolování obou souborů je na standardní výstup vytisknut přehled nalezených chyb. Pokud soubory žádnou chybu neobsahují, je vytisknut prázdný výpis.

Způsob kontroly

Nejdříve je kontrolováno, zda určitý typ pravidla (ACL nebo směrovací pravidlo) obsahuje určitý počet elementů - pro ACL pravidla je to 5 (zdrojová IP adresa, cílová IP adresa, zdrojový port, cílový port, maska protokolu), pro směrovací pravidla 6 (stejně jako ACL + je přidán identifikátor rozhraní, kam má být paket přeposlán). Příklady pravidel jsou napsány v kapitole 4.3.

[1] Seznam podporovaných ovladačů je dostupný na http://doc.dpdk.org/guides/nics/index.html

Následuje kontrola správnosti zapsání IP adres s maskou sítě. K tomuto je využito python modulu `ipaddress`. Pokud je při volání konstruktoru `ipaddress.IPv4Network()` (resp. `ipaddress.IPv6Network()`) vyvolána výjimka, je IP adresa zapsána špatně.

Pokračuje kontrola rozsahu portů, zda se zadané rozsahy pohybují v rozmezí 0 až 65535. Nakonec se kontroluje, jestli je zadána maska protokolu a případně číslo portu k přesměrovávání paketů (poslední položka směrovacího pravidla).

Pomocné programy addRule4 a addRule6

Tyto dvě utility slouží k vytváření a vkládání nových pravidel do zadaných souborů. Jsou umístěny ve složce `tools`. Oba skripty je nutné spustit s parametrem, kterým se zadává cesta k souboru s pravidly. Parametr je zadáván ve tvaru: `--ipv4_rules SOUBOR` (resp. `--ipv6_rules SOUBOR`). Hlavním cílem vývoje těchto skriptů je minimalizace chyb v souborech s pravidly. Po vložení požadovaných pravidel je doporučeno použít utilitu `checkRules` pro zkontrolování souborů. Oba skripty jsou interaktivní a všechny požadované informace jsou získávány ze standardního vstupu. Utility získávají informace v několika krocích:

- Nejdříve se zeptají, zda nové pravidlo bude **filtrovací**, či **směrovací**.
- Následně je vznesen dotaz na zdrojovou IP adresu (nutné zadávat ve formátu IP adresa/maska).
- Po zkontrolování zdrojové IP adresy skript požaduje zadání cílové IP adresy.
- Zadání počátečního čísla rozsahu zdrojových portů
- Zadání koncového čísla rozsahu zdrojových portů s následnou kontrolou, zda nyní zadané číslo není menší než to předchozí
- Zadání počátečního čísla rozsahu cílových portů
- Zadání koncového čísla rozsahu cílových portů s následnou kontrolou, zda nyní zadané číslo není menší než to předchozí
- Zadání identifikátoru protokolu ve formátu uvedeném v kapitole 4.3
- Pokud se má jednat o směrovací pravidlo, je nyní zadáno číslo portu určeného k přeposílání
- Číslo řádku, na který bude nové pravidlo přidáno

Pokud vše proběhlo v pořádku, je skript ukončen bez jakýchkoliv problémů a dalších výpisů. Pokud během zadávání bylo něco zadáno špatně, skončí skript s chybovou hláškou.

Spouštěcí skript

Tento skript se nachází ve stejné složce jako implementovaná aplikace a nese název `run.py`. Je implementován v jazyce Python a testován s interpreterem verze 3.6. Hlavním úkolem tohoto skriptu je jednodušší spouštění programu pro filtrování a přeposílání paketů a využívá parametrů zapsaných v konfiguračním souboru.

Ještě než je spuštěna aplikace, je nutné mít požadovaná síťová rozhraní svázána s ovladačem podporovaným DPDK. Bez použití spouštěcího skriptu, je nutné takzvané *„bindování“*

provádět manuálně, např. utilitou `dpdk-devbind` dostupnou v instalačním balíku DPDK. Při využití spouštěcího skriptu, je převázání rozhraní prováděno automaticky a po skončení běhu hlavní aplikace, jsou rozhraní svázána zpět s původními ovladači kernelu. K automatickému bindování byla využita upravená verze utility `dpdk-devbind`[11], dostupná je ve složce `tools` pod názvem `devbind.py` a obsahuje jen základní funkce nutné ke změně ovladačů rozhraní.

Běh spouštěcího skriptu

Nejdříve skript zpracuje konfigurační soubor. Následně zjistí identifikátory portů, podle zadaného jména a uloží je do seznamu. Ke každému rozhraní si poznamená původní ovladač a pokusí se provést bindování s ovladačem zadaným v konfiguračním souboru. Po té vytvoří ze zbytku parametrů spouštěcí řetězec, který je použit při volání `subprocess.Popen()` a tím i spuštěna výsledná aplikace. Skript se ukončí stisknutím `CTRL+C`, po němž dojde k zaslání signálu `SIGINT` aplikaci a jejímu ukončení. Nakonec se provede svázání síťových rozhrání s původním kernelovým ovladačem a ukončení skriptu.

5.3 Rozdělení práce mezi procesy

Z důvodu dosažení vysokéno výkonu aplikace, je nutné rozdělit zpracování paketů mezi několik procesů. Jak již bylo zmíněno v návrhu aplikace, konkrétně kapitola 4.4, budou všechny procesy vykonávat stejnou práci, zahrnující přijetí paketu, porovnání s pravidly a následné zahození, či přeposlání paketu na daný interface. Procesy se budou lišit v tom, se kterými pakety budou pracovat a s jakými frontami rozhraní budou pracovat. Aplikace mapuje jeden proces na jedno jádro procesoru. Počet použitých jader se určuje pomocí pamateru pro inicializaci vrstvy EAL `-c COREMASK` nebo `-l CORELIST` (viz. 2.5). Aplikace lze také spustit pouze s jedním jádrem, to ale bude mít za následek velmi negativní vliv na výkon a rychlost zpracování aplikace.

Pro rovnoměrné rozložení přijatých paketů a jejich odeslání je nutné inicializovat počet vstupních a výstupních front. Počet jednotlivých front se odvíjí od počtu jader, která byla zadána parametrem pro vrstvu EAL. To znamená, že pokud budou například zvolena dvě jádra, budou pro každé rozhraní konfigurovány dvě vstupní a dvě výstupní fronty. Tato inicializace se provádí pouze v hlavním procesu dříve, než se spustí hlavní smyčka programu na všech jádrech.

Inicializační část hlavního procesu

Hlavní proces před spuštěním ostatních procesů provede inicializaci vrstvy EAL pomocí DPDK funkce `rte_eal_init()`, které předá přijaté parametry. Tím vrstva EAL zjistí, která další jádra budou využívána (podle masky nebo seznamu jader) a uvede je do stavu čekání. Následuje zpracování zbylých aplikačních parametrů (portmaska a cesty k souborům s pravidly). Z portmasky se vypočítá pole povolených portů, v němž jsou uloženy indexy jednotlivých interfaců. Následně proběhne inicializace ACL kontextu. Po úspěšném inicializování ACL je na řadě inicializace paměťové struktury *mempool* v prostoru velkých paměťových stránek. Až je správně inicializována paměť, je možné přejít k inicializaci jednotlivých portů a jejich front voláním funkce `port_init()`. Po úspěšném dokončení všech inicializací je spuštěna hlavní smyčka zpracování paketů na všech dostupných jádrech pro-

cesoru voláním funkce DPDK knihovny `rte_eal_mp_remote_launch(lcore_main_loop, NULL, CALL_MASTER)` , kde:

- `lcore_main_loop` je funkce, ve které probíhá filtrování a přeposílání paketů v nekonečné smyčce. Tato smyčka je ukončena při přijetí signálu `SIGINT`
- `CALL_MASTER` - tento parametr nám říká, že se zadaná funkce zavolá i na hlavním procesu. Nekonečnou smyčku uvnitř funkce `lcore_main_loop` tedy budou vykonávat úplně všechny procesy

Rozdělení paketů

Pakety jsou rozřazovány jednotlivým frontám (tedy i jádrům, jelikož každé jádro zpracovává svou frontu) dle toho, jakému patří datovému toku. Datové toky jsou určeny pěticí informací extrahovaných z hlavičky paketu, jedná se o zdrojovou IP adresu, cílovou IP adresu, zdrojový port, cílový port a transportní protokol. Rovnoměrné rozdělení paketů do front obstarává technologie RSS, jenž byla popsána již v kapitole 4.4.

Nastavení výpočtu RSS hashe se provádí při inicializaci portů ve funkci `port_init()`. Povolení RSS na vstupní fronty rozhraní se provádí nastavením struktury `rte_eth_conf` následovně:

```
struct rte_eth_conf port_conf_def = {
    .rxmode = {
        .mq_mode = ETH_MQ_RX_RSS,
        .max_rx_pkt_len = ETHER_MAX_LEN,
        .split_hdr_size = 0,
        .ignore_offload_bitfield = 1,
        .offloads = (DEV_RX_OFFLOAD_CHECKSUM |
                     DEV_RX_OFFLOAD_CRC_STRIP),
    },
    .rx_adv_conf = {
        .rss_conf = {
            .rss_key = hash_key,
            .rss_hf = ETH_RSS_PROTO_MASK
        },
    },
    .txmode = {
        .mq_mode = ETH_MQ_TX_NONE,
    }
};
```

Parametr `.mq_mode = ETH_MQ_RX_RSS` aktivuje na rozhraní pro vstupní fronty RSS technologii. Výraz `.rss_hf = ETH_RSS_PROTO_MASK` říká, že pro výpočet hash hodnoty budou využity dříve zmíněné pětice (zdrojová a cílová IP adresa, zdrojový a cílový port, transportní protokol) a to jak pro IPv4 provoz, tak i IPv6 provoz.

5.4 Inicializace ACL kontextu

Příprava ACL kontextů je rozdělena do dvou souborů:

- `parseRules.c` - zde se nachází funkce k parsování souborů s pravidly.
- `aclHelpers.c` - zde jsou funkce, které nastavují struktury a vytváří kontexty.

V úvodu funkce `init_acl()` je volána další funkce `add_rules()`, kde nejdříve dojde ke spočítání jednotlivých typů pravidel (směrovací a ACL zvlášť). Po přepočítání se alokuje paměťový prostor, jehož velikost je dána počtem pravidel jednotlivých typů, pro ukazatele na pravidla. Jakmile proběhne alokace bez problému, přichází na řadu parsování souborů s pravidly po řádcích ve funkci `parse_ipv4_rule()` resp. `parse_ipv6_rule()`. Zde jsou parsovány jednotlivé elementy pravidla a je jimi naplněna DPDK struktura `rte_acl_rule`, která reprezentuje jedno pravidlo. U směrovacího pravidla je pro uložení čísla portu, kam se má přesměrovávat, využito členu `data->userdata` této struktury.

Po úspěšném parsování obou souborů a naplnění struktury reprezentující pravidla je dvakrát volána funkce `create_acl_context()` (jednou pro IPv4 a podruhé pro IPv6), kde dochází k vytváření kontextu. Nejdříve je nutné naplnit DPDK strukturu `rte_acl_param()`, kde je potřeba nastavit jméno kontextu, velikost pravidla a maximální počet pravidel. Pomocí této struktury se vytvoří ACL kontext. Jakmile je kontext vytvořený, jsou mu přiřazena přichystaná směrovací a filtrovací pravidla funkcí `rte_acl_add_rules()`. Pokud přiřazení proběhlo bez chyby, je následně na vytvořený kontext zavolána DPDK funkce `rte_acl_build()`, která zanalyzuje množinu všech pravidel a vytvoří své vnitřní struktury (binární stromové struktury trie, popsané v kapitole 2.9). Poté jsou již kontexty vytvořeny a připraveny k použití, např. vyhledávání shody s pravidly.

5.5 Inicializace portů

Konfigurace a inicializace zvolených síťových rozhraní probíhá ve funkci `port_init()` v souboru `main.c` a vykonává ji pouze jedno jádro (*master* jádro). Nejdříve je zkontrolováno, zda je zadaný port validní - tzn. zda je číslo portu v rozsahu (0 až 32) a zda je připojen. Následně je voláním funkce `rte_eth_dev_configure()` ethernetové zařízení nakonfigurováno. Tato funkce přijímá jako parametry:

- Číslo rozhraní, které bude konfigurováno
- Počet vstupních front rozhraní
- Počet výstupních front rozhraní
- Ukazatel na strukturu `rte_eth_conf`, jejíž nastavení bylo představeno v kapitole 5.3.

Jelikož každé jádro bude zpracovávat pakety na každém rozhraní, je počet front roven počtu povolených jader a počet vstupních front je stejný jako počet výstupních front. Pro ujasnění krátký příklad - pokud by byl počet síťových rozhraní dva a povolená jádra by byla také dvě, tak každé rozhraní bude mít inicializovány dvě vstupní a dvě výstupní fronty. Procesor s ID 0 bude zpracovávat fronty 0 a procesor s ID 1 bude pracovat s frontami číslo 1 každého rozhraní.

Následně je nutné nastavit fronty samotné. Nastavení vstupní fronty je provedeno DPDK funkcí `rte_eth_rx_queue_setup()`. Tato funkce alokuje souvislý paměťový blok pro 1024 příchozích deskriptorů a každý tento deskriptor inicializuje v síťovém bufferu alokovaném ve struktuře `rte_mempool`. Obdobně je nastavena fronta výstupní pomocí funkce `rte_eth_tx_queue_setup()`. Rozdíl je v tom, že se zde neuvádí buffer mempool.

Po nastavení portu a jeho vstupních a výstupních front, je síťovému rozhraní povoleno přijímání paketů v promiskuitním režimu. Tím je umožněno zachytávat i takovou síťovou komunikaci, která není primárně určena pro dané síťové rozhraní. Nakonec je zařízení nastartováno funkcí `rte_eth_dev_start()`.

5.6 Společná část procesů

Po výše popsaných inicializačních částech jsou všechna ostatní jádra probuzena dříve popsanou funkcí `rte_eal_mp_remote_launch()` a všechna přistoupí k vykonávání funkce `lcore_main_loop()`, jež je umístěna v souboru `main.c`.

Nejdříve si každé jádro alokuje a inicializuje pole výstupních bufferů `tx_buffer` (pro každý port je jeden), které budou sloužit k ukládání paketů, jež budou posléze vloženy do výstupní fronty rozhraní. Pak jádro vstupuje do „nekonečné“ smyčky, kde je vykonáváno přijímání paketů, klasifikace a následné zahození nebo vložení paketů do výstupních front. Popis těchto fází následuje v dalších podkapitolách.

5.7 Přijetí a klasifikace paketů

V rámci nekonečné smyčky probíhá další cyklus iterující přes všechna čísla portů. V každé iteraci je nejdříve získána dávka paketů pomocí funkce `rte_eth_rx_burst(port_id, queue_id, packet_burst, BURST_SIZE)`, kde:

- `port_id` je číslo síťového rozhraní, ze kterého budou přijaty pakety.
- `queue_id` je číslo fronty, ze které budou čteny pakety. Číslo fronty v naší aplikaci odpovídá číslu jádra.
- `packet_burst` je ukazatel na pole struktur typu `rte_mbuf`. Jeden prvek tohoto pole pak představuje jeden paket.
- `BURST_SIZE` udává maximální počet paketů k načtení. V naší aplikaci je rovno 32.

Tato funkce vrací počet skutečně načtených paketů, což je počet ukazatelů do `rte_mbuf` struktury. Po načtení paketů přichází přichystání dat k následnému vyhodnocení nad pravidly. U IPv4 paketů toto přichystání spočívá v kontrole, zda je přijatý paket validní. Kontrola probíhá dle sekce 5.2.2 v RFC 1812. Tento dokument říká, že než router může zpracovat paket, musí provést základní kontroly platnosti IP hlavičky, aby bylo zajištěno, že je hlavička smysluplná. Pokud některá z kontrol selže, je paket zahozen [1]. Nad hlavičkou paketu jsou provedeny následující kontroly:

- Délka paketu musí být dostatečně velká, aby udržovala minimální délku platného IP datagramu (20 bajtů).
- Kontrolní součet musí být správný (tahle kontrola je prováděna hardwarem).
- Číslo IP verze musí být 4.
- Pole délky hlavičky IP musí být dsotatečně velké, aby obsahovalo délku IP datagramu.
- Pole celkové délky IP musí být dostatečně velké, aby mohlo být uloženo v hlavičce IP datagramu, jehož délka je uvedena v poli délky hlavičky IP.

Pokud proběhla kontrola IPv4 paketu bez chyby, jsou data tohoto paketu uložena do struktury a pokračuje se dále. U IPv6 paketů neprobíhá žádná kontrola hlavičky a data jsou uložena do struktury ihned. Pro účely uchovávání informací o přijatých paketech slouží struktura:

```
struct acl_search_t {
    const uint8_t *data_ipv4[BURST_SIZE];
    struct rte_mbuf *m_ipv4[BURST_SIZE];
    uint32_t res_ipv4[BURST_SIZE];
    int num_ipv4;

    const uint8_t *data_ipv6[BURST_SIZE];
    struct rte_mbuf *m_ipv6[BURST_SIZE];
    uint32_t res_ipv6[BURST_SIZE];
    int num_ipv6;
};
```

Jejíž členové mají následující význam:

- `data_ipv*` - pole ukazatelů na data uložená ve struktuře `rte_mbuf`
- `m_ipv*` - představuje načtené a zvalidované (v případě IPv4) pakety
- `res_ipv*` - pole, které bude použito při klasifikaci. Zde budou uloženy výsledky pro jednotlivé pakety
- `num_ipv*` - udává počet načtených a validovaných paketů

Jakmile jsou data všech načtených a validovaných paketů připraveny, je zavolána DPDK funkce `rte_acl_classify()`, která přijímá parametry:

- ACL kontext, ve kterém se bude vyhledávat
- Pole ukazatelů na buffer pro vyhledávání
- Pole vyhledaných výsledků
- Počet elementů v poli bufferů
- Počet maximálních možných shod pro každý buffer. V aplikaci je defaultně nastaven na 1.

Výše uvedená funkce provede vyhledání odpovídajícího pravidla v kontextu pro všechna vstupní data. Pokud došlo ke shodě s pravidlem, je v poli výsledků (`res_ipv*`) konkrétního paketu uložena hodnota nastavená v `userdata` pravidla.

5.8 Odeslání paketů

Po klasifikaci a nastavení struktury přichází na řadu případné odeslání paketů. Ve funkci `send_one_packet()` se nejdříve kontroluje, zda hodnota výsledku `res` není rovna 0. To by totiž znamenalo, že došlo ke shodě s filtrovacím pravidlem. Pokud tomu tak je, je ze paměti struktury `rte_mbuf` tento paket uvolněn. Pokud došlo ke shodě se směrovacím pravidlem,

je paket pomocí DPDK funkce `rte_eth_tx_buffer()` vložen do výstupního bufferu pro budoucí přenos na konkrétním portu a jeho výstupní frontě.

V rámci nekonečného cyklu pak zhruba každých 100ms probíhá volání DPDK funkce `rte_eth_tx_buffer_flush()`. To způsobí explicitní odeslání paketů, které byly dříve bufferovány funkcí `rte_eth_tx_buffer()`. Funkce vrací počet úspěšně odeslaných paketů síťové kartě a pro neposlané pakety volá chybový callback. Pokud není callback nastaven, pak defaulní jednoduše uvolní neposlané pakety zpět vlastnícímu mempoolu.

Kapitola 6

Testování a měření výkonnosti

Tato kapitola se věnuje testování a měření výkonnosti s následným porovnáním výsledků se standardním firewallem `iptables`, který byl popsán v kapitole 3. K účelům testování a měření bylo poskytnuto testovací prostředí produktů NETX, vyvíjených spolkem NetX R&D z.s.[1]

6.1 Úvod do prostředí

Produkty NETX jsou založeny na vysoce výkonné a open-source směrovací a bezpečnostní platformě. Operační systém je lehce poupravený GNU/LINUX, což umožňuje jednoduchou rozšířitelnost a přizpůsobení různým síťovým úlohám.

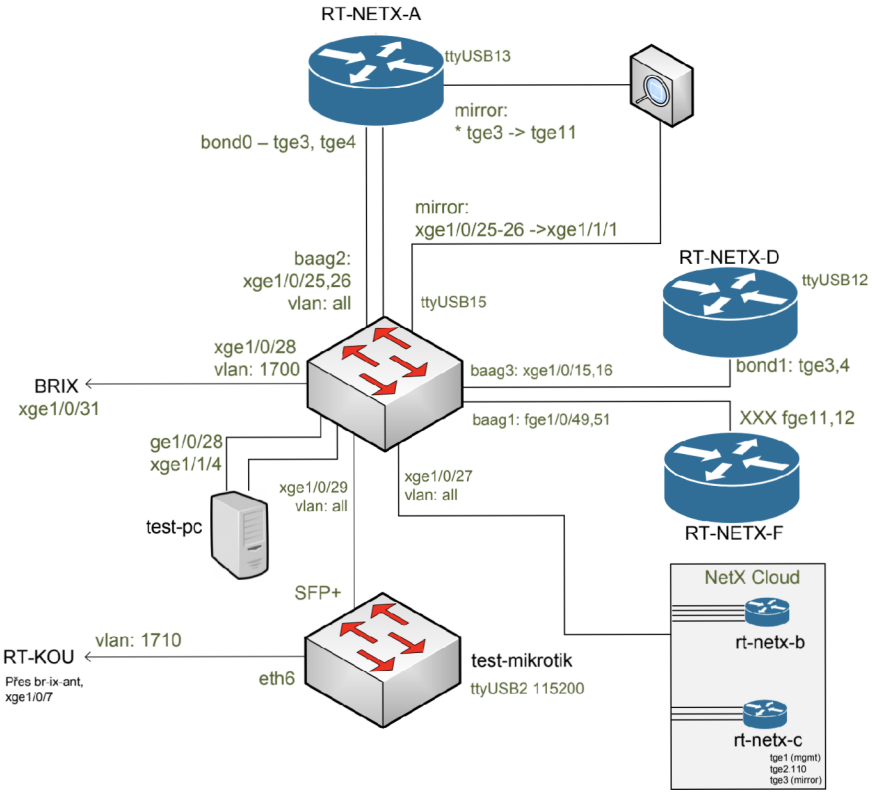
Topologie testovacího prostředí je zobrazena na obrázku 6.1. Jako experimentální router byl zvolen stroj s názvem `rt-netx-f`. Generované pakety, uloženy v pcap souborech, byly zasílány ze zařízení `test-pc` pomocí utility `pfsend` frameworku `PF_RING`[2]. Hlavní výhodou této utility je, že si lze parametry navolit, jakou rychlostí se mají pakety zasílat (pakety za sekundu nebo Gbity za sekundu) a kolik paketů se má celkem zaslat. Nicméně maximální rychlost, jaké bylo možné dosáhnout bylo 10Gbps, neboť `test-pc` bylo připojeno 10Gb portem. Zkušební router se sestával z následující konfigurace:

- **CPU** - Intel(R) Xeon(R) CPU D-1587 @ 1.70GHz (16 jader, 32 vláken)
- **RAM** - 16GB (4x 4GB 2133MHz DDR4)
- síťová rozhraní:
  - 2x Ethernet Connection X552 10 GbE SFP+ 15ac
  - 2x Ethernet Controller XL710 for 40GbE QSFP+ 1583
  - 2x I350 Gigabit Network Connection 1521
- kernel verze: 5.0.6
- `gcc` verze 4.8.5
- `python` verze 3.6
- `DPDK` framework ve verzi 18.05

[1] Webové stránky spolku `https://netx.as`

[2] `https://www.ntop.org/products/packet-capture/pf_ring/`

Následující podkapitoly se budou věnovat výkonnostním testům prováděných právě na tomto stroji.



Obrázek 6.1: Testovací topologie

6.2 Rychlost načtení pravidel

Tento test se věnuje rychlosti načítání pravidel aplikace s využitím DPDK a iptables. Bylo použito několik testovacích sad pravidel:

- 1 000 IPv4 (pouze) pravidel,
- 10 000 IPv4 (pouze) pravidel,
- 100 000 IPv4 (pouze) pravidel,
- 1 000 IPv6 (pouze) pravidel,
- 10 000 IPv6 (pouze) pravidel,
- 100 000 IPv6 (pouze) pravidel,
- 500 IPv4 + 500 IPv6 pravidel,
- 5 000 IPv4 + 5 000 IPv6 pravidel,

- 50 0000 IPv4 + 50 0000 IPv6 pravidel

Pro každou sadu bylo provedeno měření desetkrát a následně vypočtena průměrná hodnota. K načítání pravidel do standardního firewallu iptables, bylo využito utility `ipset`[3], což nám umožní uložit spoustu IP adres do jedné hash tabulky. Vyhledávání a porovnávání paketů s pravidly je pak mnohem rychlejší, než když iptables prochází všechna svá pravidla sekvenčně.

Způsob vytváření pravidel pomocí ipset

Nejdříve se vytvoří pojmenovaná sada, do níž se následně mohou vkládat jednotlivé IP adresy:

```
ipset create test_set hash:ip
ipset add test_set 192.168.1.1
```

Pokud jsou v sadě všechny potřebné IP adresy, přidá se do iptables pravidla příkazem:

```
iptables -A INPUT -m set --match-set test_set src -j DROP
```

Tím vznikne pravidlo, které porovnává adresu příchozích paketů se sadou `ipset` a pokud se v ní adresa nachází, je paket zahozen.

Během testování s velkým množstvím IP adres bylo zjištěno, že přidávat např. 10000 ip adres pomocí `ipset add` není příliš efektivní (naplnění sady trvalo několik desítek vteřin). Proto byl zvolen rychlejší způsob vytváření sad pomocí `ipset restore`, kdy jsou všechny adresy uloženy v určitém formátu v souboru. Soubor (nazývající se test.save) může vypadat například takto:

```
create test_500 hash:ip family inet hashsize 1024 maxelem 65536
add test_500 177.190.221.51
```

Tento výpis nám říká, že bude vytvořena sada s názvem `test_500`, do níž je vložena IP adresa `177.190.221.51`. Pro vytvoření je ještě nutné zavolat příkaz `ipset restore < test.save`. Čímž vznikne výše zmíněná sada a naplní se adresami. Tento postup je nespočetněkrát rychlejší a tudíž i lépe porovnání schopná s naší aplikací.

Sada 1 000 pravidel

Pravidla do iptables byla nahrána pomocí `ipset restore`, což bylo popsáno výše. Pravidla pro aplikaci byla ve formátu:

```
R192.168.1.1/32 0.0.0.0/0 0:65535 0:65535 0x0/0x0 0
```

pro IPv4 adresy, a pro IPv6 adresy následovně:

```
R2001:db8:face:b00c:1234:5678:9012:4555/128 0.0.0.0/0 0:65535 0:65535 \
0x0/0x0 0
```

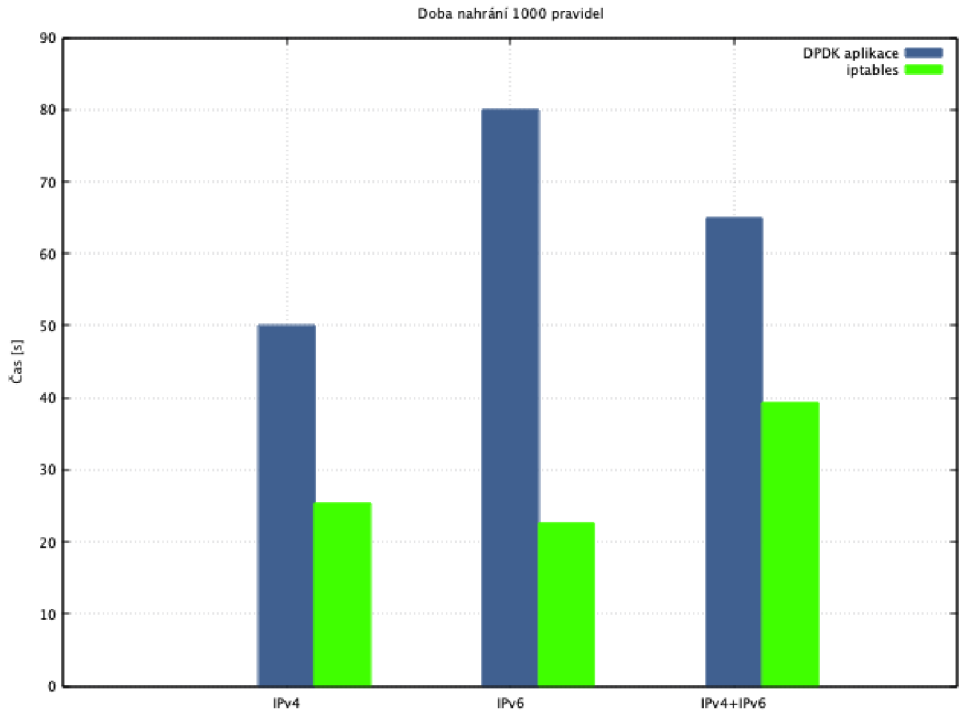
kde se pro zjednodušení měnila pouze část zdrojové IP adresy. Byly naměřeny následující hodnoty (3. sloupeček znamená 500 IPv4 + 500 IPv6 pravidel):

Z grafu 6.2 je zřejmé, že nahrávání pravidel DPDK aplikací je mnohem delší. To je zapříčiněno tím, že parsování pravidel probíhá procházením řádek po řádku souboru. A také tím, že jedno pravidlo je zapsáno složitěji, než v případě `ipset`.

[3] http://ipset.netfilter.org

	IPv4	IPv6	IPv4 + IPv6
DPDK aplikace	0.05	0.08	0.065
iptables	0.0254	0.0226	0.0394

Tabulka 6.1: Naměřené časy pro nahrání 1000 pravidel. Čas je v sekundách



Obrázek 6.2: Nahrání 1000 pravidel

Sada 10 000 pravidel

Pravidla byla nahrána stejně jako v předchozím případě. Naměřené hodnoty jsou v tabulce 6.3.

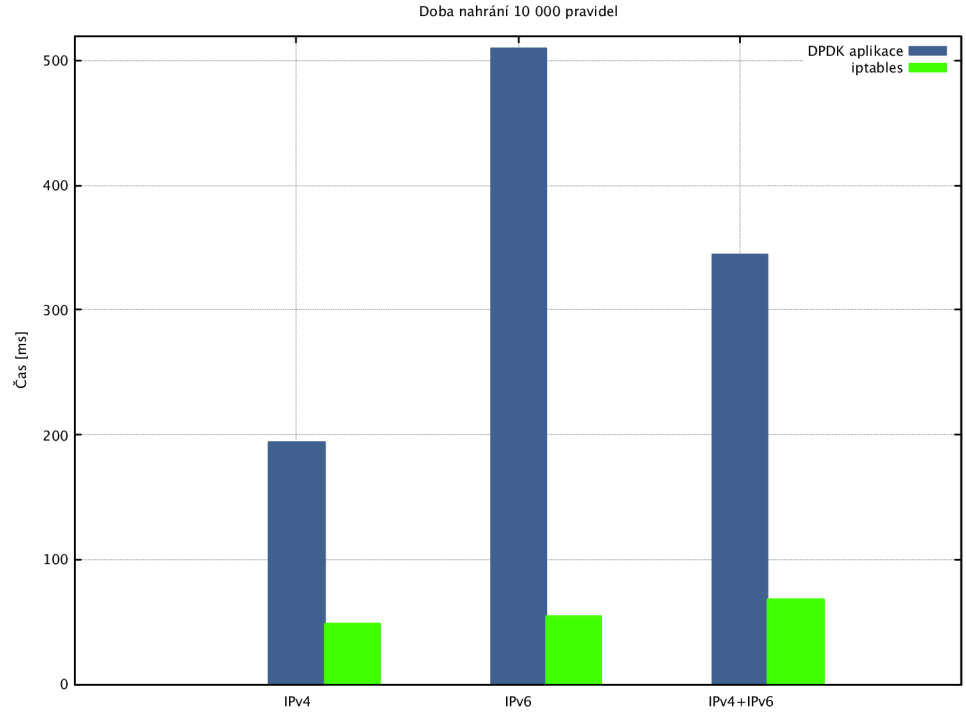
	IPv4	IPv6	IPv4 + IPv6
DPDK aplikace	0.194	0.51	0.345
iptables	0.049	0.0542	0.0676

Tabulka 6.2: Naměřené časy pro nahrání 10 000 pravidel. Čas je v sekundách

Z grafu 6.3 je vidět již markantní rozdíl, zejména u pravidel IPv6. To jest zapříčiněno tím, že tato pravidla jsou dlouhá a jejich parsování není v této aplikaci optimální.

Sada 100 000 pravidel

U této poslední sady byl obrovský rozdíl v době nahrání pravidel pomocí `ipset add` a `ipset restore`. Vytvoření sady, kdy se na každou IP adresu volalo `ipset add`, trvalo zhruba



Obrázek 6.3: Nahrání 10 000 pravidel

7 minut, oproti `ipset restore`, kterým byla sada nahrána za 0.3762 vteřiny. Proto bylo od `ipset add` úplně upuštěno a dále nevyužíváno pro potřeby testování.

V tomto testu byly naměřeny údaje zobrazené v tabulce 6.3.

	IPv4	IPv6	IPv4 + IPv6
DPDK aplikace	2.045	5.2	3.3875
iptables	0.331	0.3762	0.3618

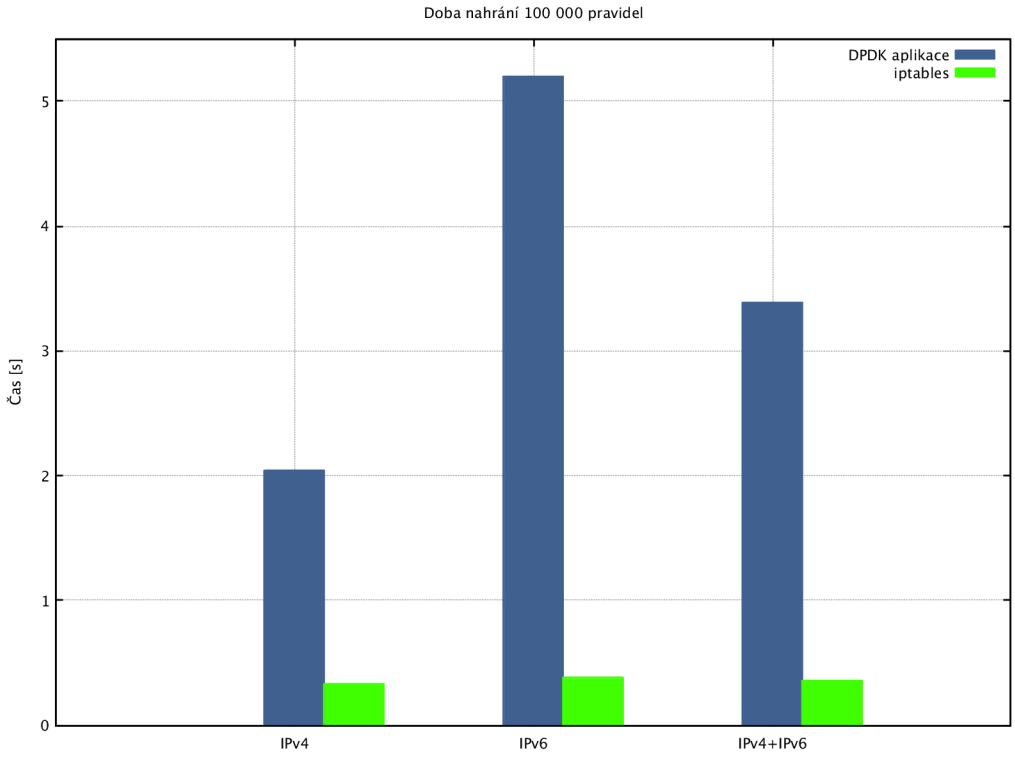
Tabulka 6.3: Naměřené časy pro nahrání 100 000 pravidel. Čas je v sekundách

6.3 Srovnání vytížení CPU

Měření vytížení procesoru při filtrování standardním firewallem iptables nebylo provedeno, jelikož by srovnání nemělo žádnou vypovídací hodnotu, a to z toho důvodu, že jakákoliv DPDK aplikace vytíží jádra procesoru, na nichž je spuštěna, na 100%.

6.4 Testy propustnosti

Cílem těchto testů bylo nastínit rozdíly v propustnostech aplikace s DPDK oproti standardnímu firewallu iptables. Testy byly prováděny se sadami zachycených paketů. Sady se lišily ve velikostech paketů, konkrétně to tedy byly sady s pakety velikosti 64, 250, 500 a 1000 bajtů. Pro každou velikost paketů byly pakety posílány na síťové rozhraní testovacího stroje.



Obrázek 6.4: Nahrání 100 000 pravidel

Pokud nedocházelo k zahazování paketů z důvodů přeplněných front, rychlost byla zvýšena a měření opakováno. Výsledná hodnota byla nejvyšší možná, při níž nebyly zahozeny žádné pakety. V připadech obou aplikací bylo nastaveno pouze jedno pravidlo. U iptables:

```
iptables -P FORWARD DROP
iptables -A FORWARD -d 100.91.56.0/24 -j ACCEPT
```

Tedy výchozí politika řetězce FORWARD byla nastavena na zahazování paketů a pravidlo, jež umožňovalo průchod paketů s danou cílovou IP adresou (všechny vygenerované pakety v sadách měly cílové adresy z rozsahu 100.91.56.0/24). U DPDK aplikace se jednalo o pravidlo:

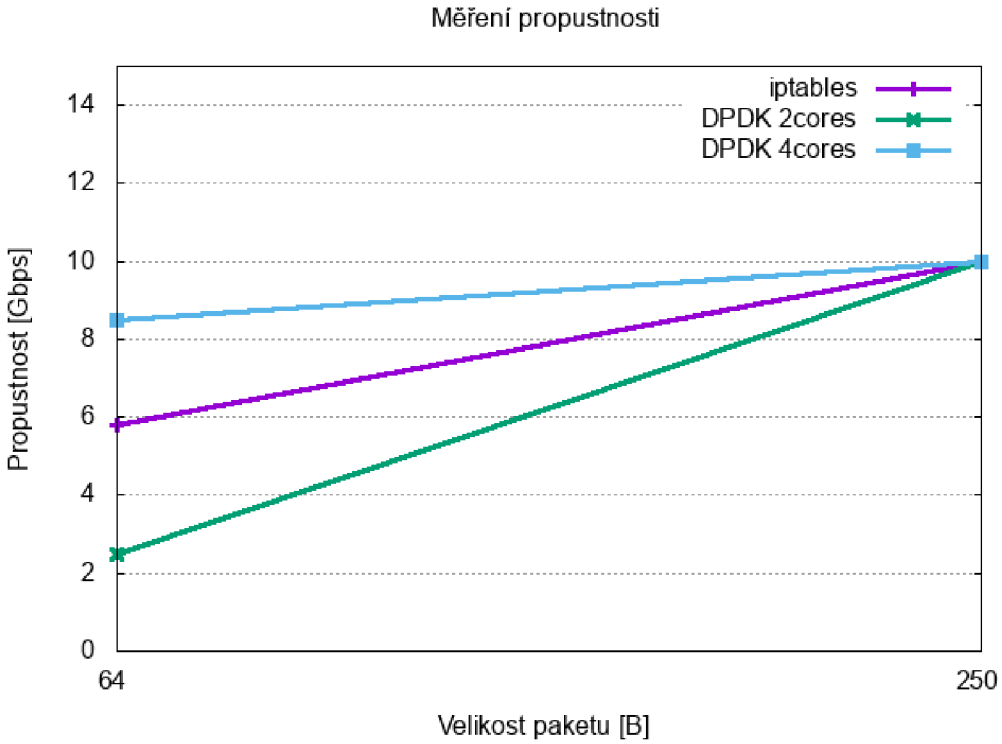
```
R0.0.0.0/0 100.91.56.0/24 0:65535 0:65535 0x0/0x0 0
```

které vykoná stejnou práci jako to u iptables. Naše aplikace byla testována dvakrát, a to jednou se dvěma a jednou se čtyřmi. Tabulka 6.4 shrnuje naměřené výsledky:

Obrázek 6.5 graficky znázorňuje rozdíly propustnosti v jednotlivých testech. Jelikož pro pakety velikosti 500 a 1000 dosahují všichni 10Gbps, nejsou v grafu zahrnuty z důvodu lepšího vyniknutí rozdílu. Je zřejmé, že oběma aplikacím tvořily problém malé pakety. Nejrychlejšího zpracování dosahovala naše aplikace se čtyřmi jádry. Naopak iptables byl schopný zpracovat maximální tok kolem 2.5Gbps.

Velikost paketů [B]	DPDK 2 jádra [Gbps]	DPDK 4 jádra [Gbps]	iptables [Gbps]
64	2.5	8.5	5.8
250	10	10	10
500	10	10	10
1000	10	10	10

Tabulka 6.4: Naměřené propustnosti



Obrázek 6.5: Měření propustnosti

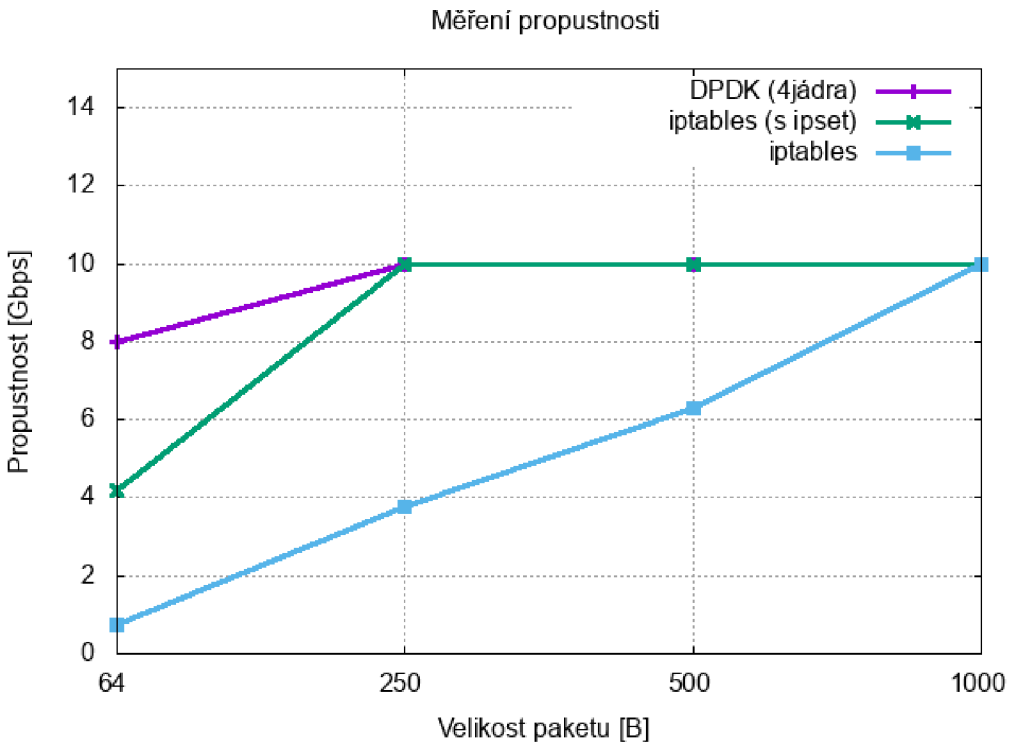
6.5 Vliv počtu pravidel na propustnost

V tomto experimentu bylo zkoumáno, zda má na propustnost nějaký vliv počet nahraných pravidel. Do naší aplikace a iptables (pomocí `ipset`) bylo nahráno 100 000 záznamů, kdy na úplný konec bylo přidáno pravidlo, ktere povolovalo/přeposílalo daný provoz. Pro zajímavost a srovnání byl také proveden test s iptables, kdy do něj bylo nahráno tisíc pravidel „starým“ stylem, tzn. každá adresa byla přidávána do seznamu příkazem `iptables -A FORWARD -d IP_ADRESA -j DROP` a nakonec bylo opět přidáno pravidlo povolující předgenerovaný provoz. Výsledky měření jsou zaznamenány v tabulce 6.5.

Z grafu na obrázku 6.6 je zřejmý mírný pokles propustnosti na velmi malých paketech. Také je vidět, že používání iptables dohromady s `ipset` má své odůvodnění, neboť iptables v jeho klasickém provedení nedosahoval takové propustnosti jako naše DPDK aplikace nebo iptables s `ipset`.

Velikost paketů [B]	DPDK (4jádra)]	iptables s ipset [Gbps]	iptables (seznam) [Gbps]
64	8	4.2	0.74
250	10	10	3.77
500	10	10	6.3
1000	10	10	10

Tabulka 6.5: Naměřené časy pro nahrání 10 000 pravidel. Čas je v sekundách



Obrázek 6.6: Měření propustnosti s vlivem pravidel

Kapitola 7

Závěr

Cílem této diplomové práce bylo nastudování knihovny DPDK, návrh aplikace pro filtrování a přeposílání vybraných paketů, která měla být následné implementována, otestována a srovnána se standardním firewallem iptables. Návrh vychází z nastudovaných komponent DPDK knihoven, zejména z ACL knihovny. Bylo také navrženo několik podpůrných programů, které usnadňují práci se soubory, kde jsou uložena jednotlivá pravidla. Navržená aplikace a utility byly úspěšně implementovány. Aplikace byla testována a výsledky testování porovnány s výsledky standardního firewallu iptables.

V kapitole 2 byly představeny základní kameny frameworku DPDK a rozdíly mezi zpracováním paketu v Linuxu s využitím a bez využití DPDK. Následující kapitola 3 se věnovala stručnému představení standardního firewallu iptables, se kterým byla výsledná aplikace porovnávána při testování v kapitole 6. Kapitoly 4 a 5 se věnovaly návrhu a následné implementaci aplikace.

Z výsledků kapitol 6.4 a 6.5 je patrné, že implementovaná aplikace dosahuje lepší propustnosti při čtyřech jádrech, než iptables. Srovnávací testování nebylo na více jádrech spouštěno, neboť nebylo možné otestovat propustnost vyšší než 10Gbps (generátor provozu byl připojen pouze 10Gbps portem). Na druhou stranu je nutné zmínit, že z výsledků kapitoly 6.2 vychází nutnost optimalizace načítání pravidel (možná i reprezentace pravidel), jelikož v některých případech zaostává za iptables téměř 10x. Tato optimalizace by mohla být zajímavé rozšíření do budoucnosti.

Literatura

[1] Baker, F.: Requirements for IP Version 4 Routers [online]. RFC 1812, červen 1995, [cit. 2019-05-15].
URL https://tools.ietf.org/html/rfc1812

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

[3] DPDK: *2. System Requirements.* [Online; navštíveno 20.12.2018].
URL https://doc.dpdk.org/guides/linux_gsg/sys_reqs.html

[4] DPDK: *42. Packet Classification and Access Control* . [Online; navštíveno 2.1.2019].
URL
https://doc.dpdk.org/guides/prog_guide/packet_classif_access_ctrl.html

[5] DPDK: *5. Ring Library* . [Online; navštíveno 30.12.2018].
URL https://doc.dpdk.org/guides/prog_guide/ring_lib.html

[6] DPDK: *6. Compiling and Running Sample Applications* . [Online; navštíveno 20.12.2018].
URL http://doc.dpdk.org/guides/linux_gsg/build_sample_apps.html

[7] DPDK: *7. EAL parameters* . [Online; navštíveno 20.12.2018].
URL https://doc.dpdk.org/guides/linux_gsg/linux_eal_parameters.html

[8] DPDK: *7. Mbuf Library* . [Online; navštíveno 30.12.2018].
URL https://doc.dpdk.org/guides/prog_guide/mbuf_lib.html

[9] DPDK: *8. Poll Mode Driver.* [Online; navštíveno 20.12.2018].
URL https://doc.dpdk.org/guides/prog_guide/poll_mode_drv.html

[10] DPDK: *About DPDK.* [Online; navštíveno 17.11.2018].
URL https://www.dpdk.org/about/

[11] DPDK: *DPDK Tools User Guides: 4. dpdk-devbind Application.* [Online; navštíveno 1.5.2019].
URL https://doc.dpdk.org/guides/tools/devbind.html

[12] DPDK: *Programmers Guide: 2. Overview.* [Online; navštíveno 17.11.2018].
URL http://doc.dpdk.org/guides/prog_guide/overview.html

[13] DPDK: *Programmers Guide: 3. Environment Abstraction Layer.* [Online; navštíveno 20.11.2018].
URL https://doc.dpdk.org/guides/prog_guide/env_abstraction_layer.html

[14] Herbert, T.; de Bruijn, W.: *Scaling in the Linux Networking Stack*. [Online; navštíveno 8.1.2019].
URL https://www.kernel.org/doc/Documentation/networking/scaling.txt

[15] KernelTalks: *What is huge pages in Linux*. [Online; navštíveno 20.12.2018].
URL https://kerneltalks.com/services/what-is-huge-pages-in-linux/

[16] Matoušek, P.: *Síťové aplikace a jejich architektura*. Brno: VUTIUM, první vydání, 2014, ISBN 978-80-214-3766-1.

[17] Piat, F.; aj.: *Hugepages*. [Online; navštíveno 20.12.2018].
URL https://wiki.debian.org/Hugepages#documentation

[18] Purdy, G. N.: *Linux iptables Pocket Reference*. O'Reilly Media, Inc., 2004, ISBN 0-596-00569-5.

[19] Saige, G.: *DPDK Design Tips (Part 1 - RSS)*. [Online; navštíveno 8.1.2019].
URL http://galsagie.github.io/2015/02/26/dpdk-tips-1/

[20] Yemelianov, A.: *Introduction to DPDK: Architecture and Principles*. [Online; navštíveno 2.01.2019].
URL
https://blog.selectel.com/introduction-dpdk-architecture-principles/

Příloha A

Obsah CD

- Text diplomové práce ve formátu PDF, včetně zdrojových souborů této zprávy pro sázecí systém LaTeX
- Zdrojové soubory knihovny DPDK a vytvářené aplikace
- Makefile pro překlad zdrojových kódů
- soubor README s návodem na instalaci
- ukázky souborů s pravidly
- Soubory s pravidly využité při testování