VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

OPTIMALIZACE SÍŤOVÝCH ÚLOH

NETWORK TASKS OPTIMALIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JAN DRAŽIL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN VIKTORIN**
SUPERVISOR

BRNO 2016

Zadání diplomové práce/18340/2015/xdrazi00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Dražil Jan, Bc.**
Obor: Počítačové a vestavěné systémy
Téma: **Optimalizace síťových úloh**
Network Tasks Optimalization
Kategorie: Počítačová architektura

Pokyny:

1. Nastudujte vybrané síťové úlohy, které je potřeba řešit v rámci sítě poskytovatelů Internetu (např. NAT, firewall, apod.).
2. Navrhněte sadu testů a otestujte několik na fakultě dostupných zařízení na výkonost v těchto síťových úlohách.
3. Navrhněte optimalizace pro zlepšení výkonu zařízení a navržené optimalizace implementujte.
4. Proveďte znovu testy na výkonost v daných úlohách po provedení optimalizací.
5. Diskutujte dosažené výsledky.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 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: **Viktorin Jan, Ing.**, UPSY FIT VUT
Konzultant: Hájek Josef, Ing., UITS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

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

L.S.

doc. Ing. Zdeněk Kotásek, CSc.
vedoucí ústavu

Abstrakt

V dnešní době, kdy se blížíme k úplnému vyčerpání veřejných IPv4 adres, se spoléháme na techniky, které kompletní vyčerpání alespoň oddálí. Jednou z těchto technik je překlad síťových adres. Internetoví poskytovatelé vyžadují od zařízení provádějících překlad síťových adres (NAT) nejvyšší možnou propustnost. Tato práce porovnává aplikaci NAT DPDK, která je založena na frameworku DPDK, s volně dostupnými implementacemi překladu stových adres. Součástí této práce je také rozšíření NAT DPDK o podporu Application-Level Gateway.

Abstract

Nowdays, when we are running out of public IPv4 addresses, we rely on techniques that at least postpone their complete exhaustion. One of these techniques is a network address translation (NAT). Internet providers require the highest possible bandwidth from devices that perform this task. This thesis compares NAT DPDK, built on top of DPDK framework, with freely available alternatives. This work also extends NAT DPDK with Application-Level Gateway support.

Klíčová slova

NAT, překlad síťových adres, porovnání výkonosti mikroprocoserů, vyčerpání IPv4 adres, DPDK, ALG, FTP, SIP, IRC

Keywords

NAT, network adress translation, microprocesor performance compare, depletion of IPv4 addresses, DPDK, ALG, FTP, SIP, IRC

Citace

DRAŽIL, Jan. *Optimalizace síťových úloh*. Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Viktorin Jan.

Optimalizace síťových úloh

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Jana Viktorina

. . . . . .
Jan Dražil
23. května 2016

Poděkování

Na tomto místě bych rád poděkoval Ing. Janu Viktorinovi za ochotu a čas, který mi při vedení mé práce věnoval.

Obsah

1 Úvod **3**

2 Překlad síťových adres **4**
2.1 Základní NAT 6
2.1.1 Restriktivita základního NATu 9
2.2 NAPT 10

3 Měření propustnosti implementace překladu síťových adres v Linuxu **14**
3.1 Generátor paketů 15
3.2 Popis měření 15
3.3 Výsledky měření 17

4 Překlad síťových adres nad knihovnou DPDK **22**
4.1 DPDK 22
4.1.1 EAL 22
4.1.2 Ring 23
4.1.3 Mempool 23
4.1.4 Message buffer 24
4.1.5 Hashovací tabulka 24
4.1.6 Ovladače síťových karet 25
4.1.7 KNI 25
4.2 Překlad síťových adres s DPDK 26
4.2.1 Základní architektura aplikace NAT DPDK 26
4.2.2 Distributor 26
4.2.3 Implementace asociační tabulky v NAT DPDK 27
4.2.4 Worker 29

5 Zpracování aplikačních protokolů v NAT DPDK **35**
5.1 Rozhraní pro zpracování aplikačních protokolů 35
5.2 TCP mangling 37
5.3 FTP 41
5.4 SIP 44
5.5 IRC DCC 49

6 Testování a měření **51**
6.1 Testování ALG pro FTP 51
6.2 Testování ALG pro SIP 52
6.3 Testování ALG pro IRC 52

6.4 Měření 52

7 Závěr **55**

Literatura **56**

Přílohy **58**
Seznam příloh 59

A Výsledky všech provedených měření **60**
A.1 AMD Opteron 6376 10Gbps – Netfilter 60
A.2 Intel Xeon E5-2620 10Gbps – Netfilter 64
A.3 Intel i3-6300 1Gbps – Netfilter 68
A.4 Intel i7-6700 1Gbps – Netfilter 71
A.5 Intel Xeon E5-2620 10Gbps – NAT DPDK 72

Kapitola 1

Úvod

V dnešní době je při návrhu počítačové sítě potřeba plánovat, jak se budou využívat veřejné IPv4 adresy, a to důkladněji než tomu bylo v minulých letech. Důvodem je jejich nedostatek. Organizace IANA[1] totiž všechny přidělitelné bloky veřejných IP adres rozdala regionálním správcům v roce 2011. Regionálním správcům veřejné IP adresy začínají docházet také, a proto přešli na přísnější režim udělování IP adres lokálním poskytovatelům Internetu. Lokální poskytovatelé, pokud chtějí zachovat svoji IPv4 infrastrukturu, musejí hledat řešení jak malé množství IPv4 adres přidělit velkému počtu klientů.

Internetoví poskytovatelé mohou problém vyčerpání řešit několika způsoby. Jedním z nich je nakoupit IPv4 adresy na trhu, což je poměrně nákladné (v době psaní této práce se cena 1 veřejné IPv4 adresy pohybovala kolem 10 USD[2]). Alternativní možností je neposkytovat klientům konkrétní 1 veřejnou IPv4 adresu, ale dočasněji jim nějakou „zapůjčit“. To řeší problém pouze v případě, že veřejnou IP adresu budou v danou chvíli potřebovat jen někteří klienti. Samozřejmě stále může nastat situace, kdy bude chtít používat Internet více klientů, než má internetový poskytovatel dostupných veřejných IPv4 adres. V tomto případě nezbývá nic jiného, než vice klientům zapůjčit stejnou veřejnou IPv4 adresu. Zapůjčení jedné IP adresy jednomu, nebo více klientům je možné provést za pomocí technologie zvané překlad síťových adres (angl. network address translation, NAT).

Cílem této práce je popsat jak technologie překladu adres fungují. V další části se zabývá tím, jak výkonná je implementace překladu síťových adres v operačním systému Linux a do jaké míry záleží na použitém hardwaru. V následujících kapitolách je popsána knihovna DPDK a implementace překladu síťových adres za využití této knihovny. Poté jé vysvětleno, jakým způsobem je tato implementaci rozšířena o podporu protokolů, které nejsou implicitně překlad adres podporovány. Na závěr je provedeno měření, jakých propustností implementace dosahuje.

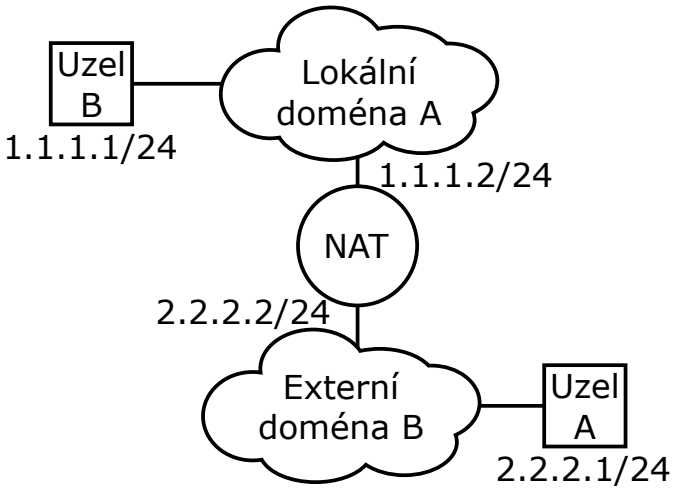
[1] http://www.iana.org

[2] Zdroj: http://ipv4marketgroup.com/broker-services/buy/

Kapitola 2

Překlad síťových adres

Uvažujme, že počítačová síť (dále jen síť) sestává ze dvou částí, které se nazývají lokální doména a externí doména (další používané pojmy jsou vnitřní a vnější doména, případně privátní a veřejná doména). Předpokládejme, že uzel[1] je vždy součástí jen jedné domény. Výjimku tvoří hraniční uzel, který budeme nazývat NAT[2]. Každý uzel má v rámci své domény přidělenou alespoň jednu unikátní adresu, kde NAT má přidělenou alespoň jednu adresu z každé domény, jejíchž je členem (znázornění na obrázku 2.1).



Obrázek 2.1: Zapojení NATu

Rozlišujme mezi paketovým tokem a relačním tokem. Paketový tok v případě TCP/IP sítí je množinou paketů, které projdou za určitý čas určitým bodem v síti a mají stejné atributy, jako jsou např. zdrojové/cílové IP adresy, zdrojové/cílové porty, nebo použité komunikační protokoly [20]. Zatímco paketový tok běžně identifikuje pakety pouze v jednom směru (od odesílatele k příjemci), relační tok identifikuje pakety, které patří k dané relaci. Jinak řečeno relační tok je vytvořen ve chvíli, kdy uzel A začne komunikovat s uzlem B a následně obsahuje pakety, které odeslal uzel A uzlu B, ale i pakety odeslané v obráceném směru.

[1] Koncové zařízení v počítačové síti. Např. PC, notebook, mobilní telefon apod.
[2] Často se pojem NAT používá jako označení počítače, který provádí překlad adres.

Funkce, kterou provádí NAT, se nazývá transparentní směrování. Transparentní směrování se od směrování, které provádějí běžné síťové směrovače, liší v tom, že běžné směrovače provádějí směrování jen v rámci jedné domény. Zatímco pro běžné efektivní směrování dostačují informace o paketovém toku, pro transparentní směrování jsou potřeba informace o relačním toku.

Překlad síťových adres je metoda, kdy je mapována adresa z jedné domény na adresu domény druhé a zároveň je zajištěna funkce transparentního směrování mezi komunikujícími uzly [27]. Existuje více použití překladu síťových adres. Například je možné provádět překlad síťových adres, protože chceme sdílet jednu veřejnou IP adresu mezi více počítači, nebo chceme překlad provádět tak, abychom paketové toky směřující na jeden uzel rozdělili na více uzlů a vytvořili tzv. LSNAT [25].

Většina implementací překladu síťových adres vychází ze stejného konceptu. Nejdříve je NATu přidělena adresa z externí i lokální domény. O tom jaké adresy jsou přiděleny, rozhodují zpravidla správci daných domén. Následně NAT provádí transparentní směrování paketů, tak aby ve výstupní doméně mohly být korektně směrovány. Na obrázku 2.1 je znázorněn příklad, jak může být zařízení provádějící překlad adres zapojeno.

Operace prováděné při překladu síťových adres jsou následující:

Asociace adresy
: Operace je provedena ve chvíli, kdy vzniká nový relační tok mezi dvěma doménami. Lokální adresa uzlu, který navazuje komunikaci s uzlem v externí doméně, je asociována s volnou adresou, která se nachází v externí doméně. V tomto případě hovoříme o tzv. dynamické asociaci. Alternativně je také možné konkrétní asociace přímo nastavit, pak hovoříme o statické asociaci. Pro jakoukoliv následnou komunikaci uzlu směrem do externí domény a komunikaci směřující na asociovanou externí adresu z externí domény je aplikována vytvořená asociace.

Vyhledání asociace
: Při příchodu paketu na rozhraní uzlu, které provádí překlad síťových adres, je pro tento paket vyhledána asociace. Pokud je nalezena, paket je podle odpovídající asociace přeložen a odeslán. Pokud není nalezena, je provedena operace asociace adresy (viz výše). Následně se s novou asociací postupuje stejně, jako by byla úspěšně nalezena.

Uvolnění asociacované adresy
: Když je vyhodnoceno, že asociace není potřeba, proběhne její odstranění. Po odstranění asociace je možné původně asociovanou externí adresu znovu použít.

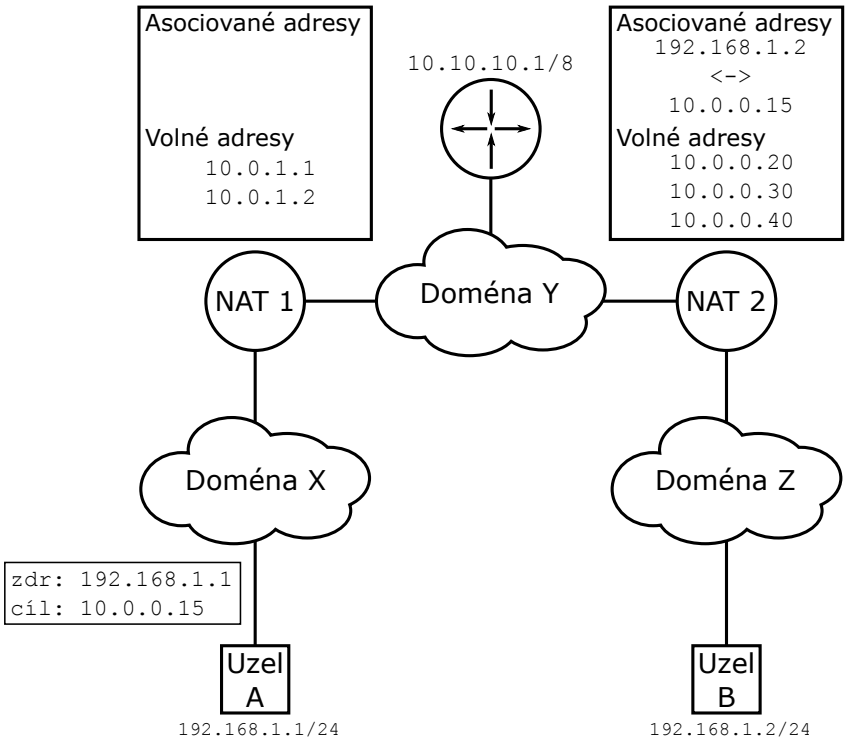
Překlad síťových adres je možné provádět na různých vrstvách referenčního modelu ISO/OSI. V této práci se však budeme zabývat pouze překladem síťových adres, který umožňuje provádět transparentní směrování v rámci architektury TCP/IP se zaměřením na IPv4. Při zasazení do kontextu předešlých pojmů se NAT, na který cílíme, používá tak, že privátní rozsah IPv4 reprezentuje lokální doménu a Internet doménu externí. Takový NAT je možné rozdělit na dva typy:

1. Základní NAT, který zasahuje pouze do IP hlaviček.
2. NAPT (Network Address Port Translation), který navíc zasahuje na 4 vrstvu referenčního modelu ISO/OSI.

Popis na jakém principu tyto dva typy pracují jsou popsány v následujících kapitolách.

2.1 Základní NAT

Základní NAT [26] má přidělenou danou množinu unikátních adres z externí domény. NAT následně provádí mapování lokálních IP adres na přidělenou množinu externích IP adres. V případě, že se v lokální doméně nachází stejný nebo menší počet uzlů než je počet přidělených externích adres, pak je možné zaručit, že ke každému uzlu bude asociována unikátní adresa z externí domény. V opačném případě bude přístup do externí domény limitován velikostí množiny přidělitelných adres z externí domény. Za výhodu základního NATu se považuje, že pokud má nějaký uzel asociovanou adresu z externí domény, pak je tento uzel dostupný z libovolného uzlu v externí doméně.

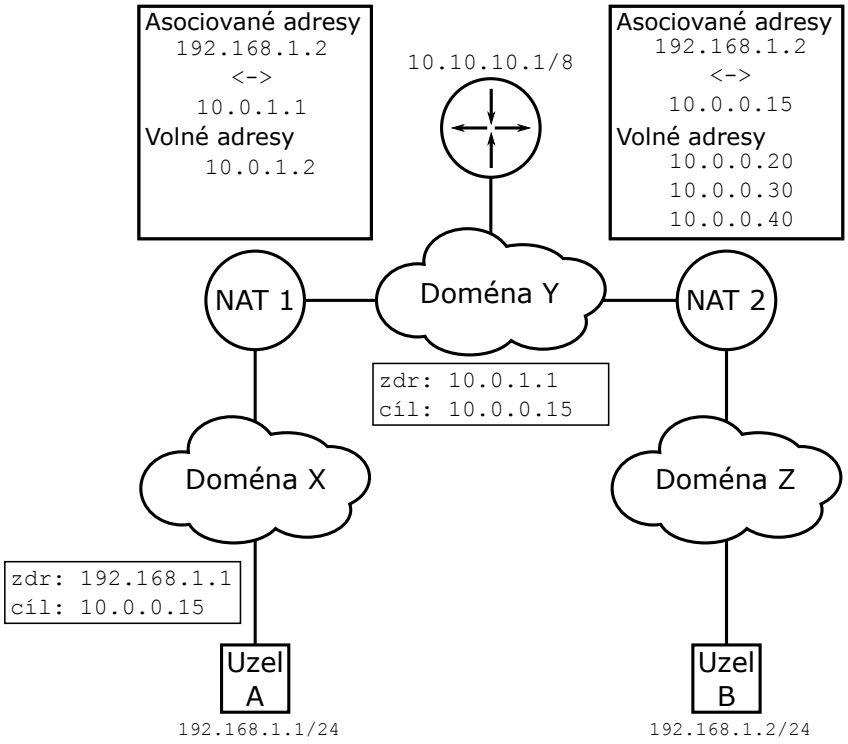


Obrázek 2.2: Příklad použití základního NATu, krok 1

Obrázek 2.2 ukazuje příklad použití základního NATu. Topologie v příkladu sestává ze 3 domén. Pokud jsou domény odděleny více než jedním NATem, pak mohou obsahovat navzájem konfliktní IP adresy. Nicméně v příkladu jsou pro přehlednost zvoleny IP adresy rozdílné.

Uzel NAT 1 může mezi doménami X a Y asociovat uzly v doméně Y s IP adresami 10.0.1.1 a 10.0.1.2. Uzel NAT 2 má mezi doménou Y a Z pro asociaci k dispozici IP adresy 10.0.0.20, 10.0.0.30 a 10.0.0.40, navíc má provedenou statickou asociaci IP adresy 192.168.1.2 na adresu 10.0.0.15. Oba uzly NAT, kromě překladu síťových adres, slouží jako směrovače.

Uzel A chce komunikovat s některou službou, která je dostupná na IP 10.0.0.15 (uživatel na uzlu A nemusí vůbec vědět, že IP 10.0.0.15 je IP uzlu NAT 2 a nikoliv cílového uzlu). Na uzlu A je vytvořen paket se zdrojovou IP adresou uzlu A a cílovou IP 10.0.0.15. Následně je paket odeslán a doručen na NAT 1.

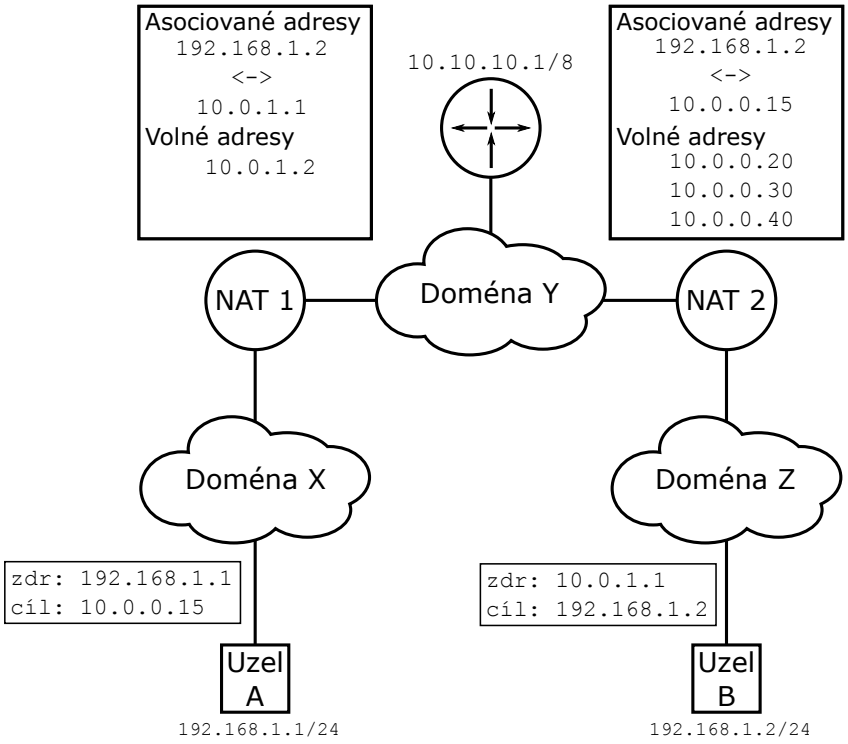


Obrázek 2.3: Příklad použití základního NATu, krok 2

Uzel NAT 1 nad paketem, který dorazil na jeho lokální rozhraní, provede transparentní směrování. Posloupnost jednotlivých akcí je následující:

1. Zkontroluje se, zda zdrojová IP adresa má asociovanou adresu z domény Y.
2. Zjistí se, že žádná adresa ještě nebyla asociována a je vybrána jedna z volných adres, které byly uzlu NAT 1 přiděleny.
   - Pokud je k dispozici více než jedna adresa pro výběr, pak jaká adresa bude vybrána, záleží na konkrétní implementaci překladu adres.
   - V příkladu byla vybrána IP adresa 10.0.1.1, která byla asociována se zdrojovou IP adresou přijatého paketu (192.168.1.1).
3. U přijatého paketu je upravena zdrojová IP adresa na adresu, se kterou je tato IP adresa asociována, konkrétně na 10.0.1.1.
4. Upravený paket je odeslán do domény Y.

Po odeslání z uzlu NAT je paket doručen na uzel NAT 2. Tato situace je zobrazena na obrázku 2.3.



Obrázek 2.4: Příklad použití základního NATu, krok 3

Na předešlém příkladu bylo provedeno transparentní směrování z lokální domény do domény externí. Nyní se bude provádět transparentní směrování v obráceném směru. Uzel NAT 2 mezi doménou Y a Z přijal paket na svém externím rozhraní a provede následující akce:

1. Zkontroluje, zda cílová IP adresa (10.0.0.15) paketu má přidělenou asociaci na adresu z lokální domény[3].

2. U příchozího paketu je zaměněna cílová IP adresa (cílová, protože se jedná o transparentní směrování z externí domény do lokální) za adresu 192.168.1.2.

3. Upravený paket je odeslán do domény Z.

V doméně Z je paket doručen uzlu B (obrázek 2.4). Pokud by byla na doručený paket vyžadována odpověď, pak by uzel B odpovídal na adresu 10.0.1.1, nikoliv 192.168.1.1, kterou ani nezná. Následné zpracování paketu s odpovědí by bylo zpracováno obdobně jako v případě komunikace od uzlu A k uzlu B, jenom v opačném pořadí.

Představme si ještě situaci, která je na obrázku 2.2, s tím rozdílem, že komunikaci by zahajoval uzel B s cílem dosáhnout uzlu A. Vzhledem k tomu, že uzel A ještě nemá asociovanou žádnou adresu z domény Y, tak uzel B ani nemůže vědět, jakou cílovou IP adresu má použít. Existují dvě situace, ve kterých by se uživatel uzlu B mohl domnívat, že IP adresu zná. V první situaci, má přístup k uzlu A a mohl by zjistit, jak je nakonfigurován. Ovšem není mu známo, že komunikace mezi uzlem A a uzlem B musí být transparentně směrována. Pak by mohl vyslat paket na IP adresu 192.168.1.1. Tento paket by neopustil

[3]V příkladu je na uzlu NAT provedena statická asociace IP 192.168.1.2 na 10.0.0.15.

doménu Z a byl by doručen buď špatnému uzlu v doméně Z, nebo by byla vrácena odpověď ICMP protokolem o nedosažitelnosti cíle.

Druhá situace nastane pokud uzel B v minulosti komunikoval s uzlem A. Uzel B si stále pamatuje, že uzel A má IP adresu 10.0.1.2, ale NAT 1 tuto IP adresu už uvolnil. Uzel B při pokusu o navázání spojení s uzlem A vyšle paket s cílovou IP adresou 10.0.1.2. Paket dorazí na NAT 2, který má dvě možnosti:

1. V ARP tabulce má záznam o IP 10.0.1.2. Provede překlad přijatého paketu a pošle ho na NAT 1.

2. Záznam v ARP tabulce nemá, tak se jej pokusí vytvořit prostřednictvím ARP protokolu. Jestli NAT 1 na ARP dotaz, který se snaží získat MAC adresu pro neasociovanou adresu, odpoví závisí na jeho konkrétní implementaci.

   - Pokud NAT 1 odpoví, pak je paket přeložen a přeposlán na NAT 1.
   - V opačném případě je paket zahozen a odesílatel je o této skutečnosti informován prostřednictvím protokolu ICMP.

V případě, že na NAT 1 je doručen paket s cílovou IP adresou, která nemá vytvořenou asociaci (v případě IP adresy 10.0.1.2 toto nastane), pak je paket zahozen a informuje se odesílatel.

Na příkladu si lze všimnout, že oba uzly NAT provádějí transparentní směrování mezi privátními rozsahy IP adres. Uzly NAT je dokonce možné řadit za sebe, například pokud by na obrázku 2.2 byl místo směrovače další uzel NAT, který by byl připojen do Internetu, pak by překlad adres na paketech z domén X a Z směřující do Internetu byl proveden dvakrát.

2.1.1 Restriktivita základního NATu

Předpokládejme statické přidělení asociace adresy (u dynamické je funkce obdobná), kde IPv4 adresa 10.10.10.10 z lokální domény má asociovanou adresu 7.7.7.7. Pokud je vyslán paket z uzlu s adresou 10.10.10.10 je na uzlu NAT přeložena na 7.7.7.7 a paket je odeslán do externí domény, vznikne tak nové spojení. Pokud však má vzniknout nové spojení z externí domény a paket je cílen pro adresu 7.7.7.7, to zda se spojení vytvoří záleží na tom, zda NAT pracuje v restriktivním módu, či ne.

Restriktivní mód Každé spojení, které je zahájeno z uzlu v lokální doméně směrem na uzel náležící v externí doméně, je zaevidováno. Například se ukládá pětice (asociovaná adresa, zdrojový L4 port, cílová IP adresa, cílový L4 port, použitý L4 protokol). Pokud na NAT dorazí paket, který má v pětici pouze prohozené zdrojové a cílové adresy, pak je tento paket přeložen a odeslán na uzel v lokální síti. Pokud na NAT dotazí paket, který není možné v evidovaných spojeních dohledat, pak je paket zahozen a to i přes fakt, že cílová adresa pakety je platná asociovaná adresa.

Nerestriktivní mód
Libovolný paket doručený na rozhraní NATu, který má v cílové adrese jednu z adres, které jsou asociovány s uzlem v lokální doméně, je přeložen a odeslán na uzel v lokální doméně.

Pokud je nasazen restriktivní NAT, pak z pohledu uzlů v externí doméně, se uzly v lokálnídoméně mohou jevit, jako by před nimi byl firewall, který umožňuje navazovat spojení

pouze z vnitřní části sítě. Obecně se však funkcionalita NATu nepovažuje (a ani se nedoporučuje považovat) za technologii, která be měla poskytovat bezpečnostní funkce, jaké má např. zmíněný firewall [5].

2.2 NAPT

Představme si situaci, kdy máme na základním NATu k dispozici jen jednu IP adresu z externí domény a v lokální doméně máme alespoň dva uzly. Pokud by jen jeden uzel v lokální doméně chtěl navázat spojení s uzlem v externí doméně, pak nenastane žádný konflikt. V případě, kdy by chtěly navázat spojení 2 uzly z lokální domény, pak by došlo ke konfliktu. Jeden uzel má asociovanou IP adresu z externí domény a k druhému není možné asociovat žádnou další adresu. Druhý uzel tedy musí počkat, dokud se externí IP adresa neuvolní a následně může komunikovat, čímž opět zamezí možnost komunikace s externí doménou uzlu prvnímu.

Výše popsaný problém řeší Network Adress Port Translation (NAPT) [26]. Na rozdíl od základního NATu, který pro pokrytí N uzlů vyžaduje alespoň N asociovatelných IP adres z externí domény, NAPT vystačí s mnohem nižším počtem asociovatelných IP adres. NAPT staví na stejných principech jako základní NAT, ale navíc do překladu adres zahrnuje také adresy ze 4 vrstvy referenčního modelu ISO/OSI (dále v textu uvažujme, že pracujeme s TCP a UDP porty).

NAPT má přiřazenou určitou množinu asociovatelných IP adres z externí domény. Při vzniku relačního toku směrem z lokální domény do externí je k zdrojové IP adrese asociována jedna externí IP adresa a zároveň ke zdrojovému TCP/UDP portu asociován port takový, aby dvojice (asociovaná IP adresa, asociovaný port) byla unikátní v rámci jednoho uzlu NAT. Tento mechanismus je schopen ke každé asociovatelné IP adrese přiřadit teoreticky až 131 072 relačních toků (65 536 pro TCP a 65 536 pro UDP). Protože porty v rozsahu o 0 do 1 024 pro TCP a UDP jsou rezervovány, tak prakticky je možné přiřadit 129 024 portů.

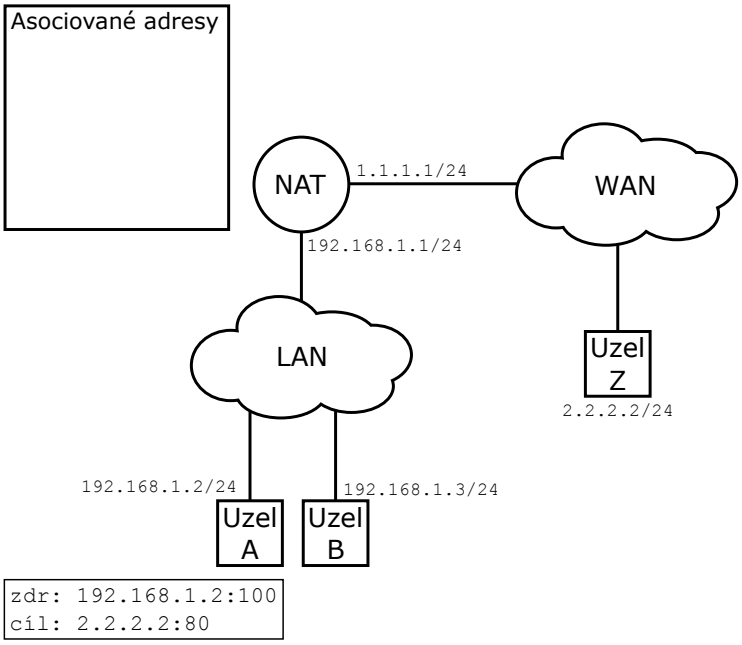
Na obrázku 2.5 je příklad nasazení NAPT. LAN tvoří lokální doménu a WAN doménu externí. V lokální doméně se nacházejí 2 uzly, které budou komunikovat s uzlem Z, který je v externí doméně. Uzly v lokální doméně mají nastavenou výchozí bránu na zařízení provozující NAT, který slouží pro překlad síťových adres a také jako směrovač. Nejdříve uzel A posílá paket uzlu Z. Důležité informace v odeslaném paketu pro překlad adres jsou:

- zdrojová IP adresa (192.168.1.2)
- zdrojový port (100)
- cílová IP adresa (2.2.2.2.2)
- cílový port (80)

Při průchodu paketu sítí v uvedeném příkladu jsou provedeny následující kroky:

1. Paket je odeslán do sítě LAN a doručen na uzel NAT[4].
2. Na uzlu NAT se pro přijatý paket zjistí, zda pro dvojici zdrojová IP adresa a zdrojový port byla v minulosti vytvořena asociace. Asociace není nalezena a tak se nová vytvoří následovně:

[4]V tomto příkladu implicitně předpokládejme, že veškerá komunikace je vedena přes protokol TCP. Pokud bychom uvažovali i UDP, pak by v záznamu o asociaci adres musel být uveden i použitý L3 protokol.



Obrázek 2.5: Příklad použití NAPT, krok 1

(a) Přidělí se externí IP adresa 1.1.1.1 (v daném příkladu není jiná dostupná).

(b) Pro správnou funkčnost NAPT je také potřeba zvolit port pro asociaci. Zde je zvolen port 21001 (volba konkrétního portu závisí na konkrétní implementaci překladu adres).

(c) Je vytvořena asociace (192.168.1.2, 100) ↔ (1.1.1.1, 21001).

3. V paketu je upravena zdrojová IP adresa na adresu 1.1.1.1, zdrojový port na port 21001.

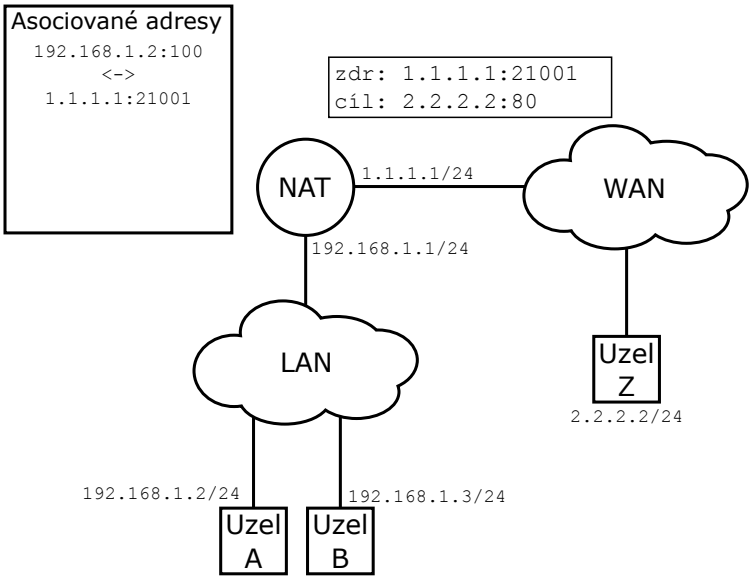
4. Paket je odeslán do sítě WAN (obrázek 2.6).

Případná odpověď od uzlu Z bude směrována na IP adresu 1.1.1.1 a port 21001. Pokud takový paket dorazí na rozhraní uzlu NAT připojenému k síti WAN, dvojice cílová IP adresa a port bude upravena na IP adresu 192.168.1.2 a cílový port 100. Stejně jako v případě základního NATu ani jeden uzel v síti pro správnou funkčnost nevyžaduje informaci o tom, že mezi komunikujícími uzly je NAPT.

Pokud uzel A bude chtít navázat nové spojení s uzlem Z, ale na jiném portu (obrázek 2.7, krok 1), pak postup bude velmi obdobný jako v minulém případě. Jenom v kroku 2b je zvolen další volný port, konkrétně 21002 a v kroku 2c je vytvořena asociace (192.168.1.2, 101) ↔ (1.1.1.1, 21002) (obrázek 2.7, krok 2). Odeslaný paket má upravené hlavičky adekvátně k novému záznamu (obrázek 2.7, krok 3).

Nyní lze vidět, že na rozdíl od základního NATu, který by měl pro uzel A pouze jednu asociaci, která by shrnovala veškeré relační toky spojené s uzlem A, má NAPT vytvořenou asociaci pro každý jeden relační tok.

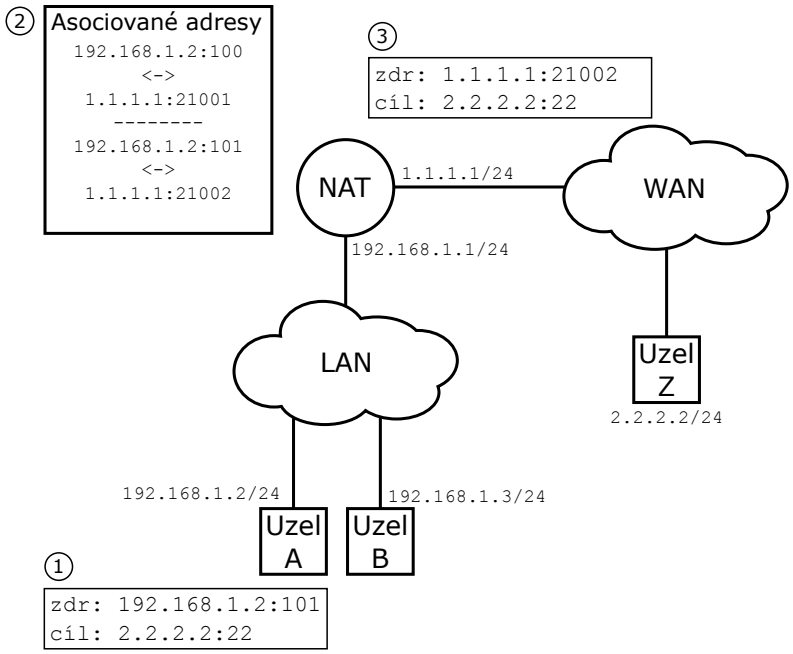
Nyní předpokládejme, že uzel B chce vytvořit spojení s uzlem Z (obrázek 2.8, krok 1). První paket v daném spojení je odeslán do sítě LAN, odkud dojde na uzel NAT. NAT provede následující operace:



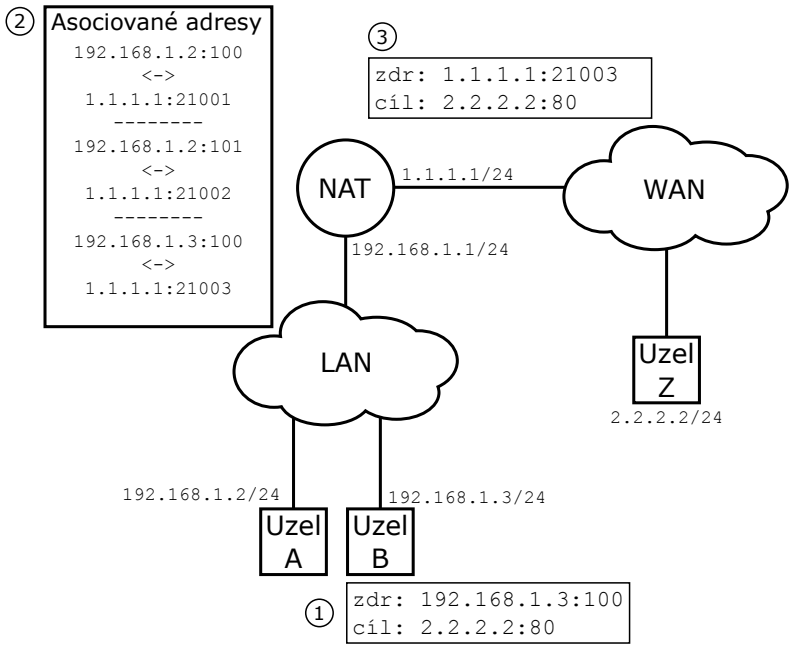
Obrázek 2.6: Příklad použití NAPT, krok 2

1. Zjistí se, zda pro dvojici zdrojová IP adresa a zdrojový port byla v minulosti vytvořena asociace. Asociace není nalezena a tak se nová vytvoří následovně:
   (a) Přidělí se externí IP adresa 1.1.1.1.
   (b) Přidělí se další volný port, konkrétně 21003.
   (c) Je vytvořena asociace (192.168.1.3, 100) ↔ (1.1.1.1, 21003) (obrázek 2.8, krok 2).
2. V paketu je upravena zdrojová IP adresa na adresu 1.1.1.1, zdrojový port na port 21003.
3. Paket je odeslán do sítě WAN (obrázek 2.8, krok 3).

Pokud by se jednalo o základní NAT, který by měl jedinou asociovatelnou adresu přiřazenou k uzlu A, pak by musel být přijatý paket zahozen a spojení mezi uzlem B a Z by nevzniklo. Také je možné si všimnout, že uzel B použil zdrojový port, který v minulosti použil uzel A. Dvojice (192.168.1.3, 100) je však v rámci sítě unikátní, a proto se nejedná o problém. K přijatému paketu je asociována nová dvojice (1.1.1.1, 21003).



Obrázek 2.7: Příklad použití NAPT, navázání dalšího spojení (čísla v kroužku označují jednotlivé kroky)



Obrázek 2.8: Příklad použití NAPT, navázání spojení z dalšího uzlu (čísla v kroužku označují jednotlivé kroky)

Kapitola 3

Měření propustnosti implementace překladu síťových adres v Linuxu

Cílem měření bylo zjistit jaká platforma je pro překlad síťových adres vhodná. Pro srovnání propustnosti překladu síťových adres byla vybrána implementace NAPT, která je součástí jádra operačního systému Linux. Tato implementace byla zvolena, protože operační systém Linux je dostupný na všech měřených platformách.

Měřené platformy jsou následující:

- AMD Opteron 6376 @ 2.3 GHz (na základní desce 4x procesor, každý po 16 jádrech), 64GB DDR3 RAM, konektivita: 2×10G + 2×1G porty
- Intel Xeon E5-2620 @ 2.00 GHz (6 jader + HT[1]), 32 GB DDR3 RAM, konektivita: 2×10G + 2×1G porty
- Intel i3-6300 @ 3,8 GHz (2 jádra + HT), 16 GB DDR4 RAM, konektivita: 1x1G port
- Intel i7-6700 @ 3,4 GHz (4 jádra + HT), 16GB DDR4 RAM, konektivita: 1×1G port

Pro srovnání se standardními desktopovými platformami bylo provedeno základní měření i na následujících zařízeních:

- OLinuXino Lime A20 (ARM Cortex-A7 Dual-Core @ 1 GHz, 512 MB DDR3 RAM), konektivita: 1×1G port
- HardKernel Odroid-C1+ (ARM Cortex-A5 Quad-core @ 1,5 GHz, 1 GB DDR3 RAM), konektivita: 1×1G port

K dispozici byla také platforma obsahující mikroprocesor ARM s technologií big.LITTLE:

- HardKernel Odroid-XU4 (ARM Cortex-A15 Quad-core @ 2 GHz, ARM Cortex-A7 Quad-core 1,4 GHz, 2 GB LPDDR3 RAM), konektivita: 1×1G port

Ovladače síťové karty u platformy HardKernel Odroid-XU4 obsahují chybu, která znemožnila tuto platformu změřit. Chyba se projevuje při vyšším zatížení síťové karty tak, že se kontinuálně zvyšuje množství spotřebované operační paměti, až dojde k jejímu úplnému vyčerpání a operační systém se s chybou ukončí.

[1] Hyper-threading

3.1 Generátor paketů

Pro Měření zátěže síťového prvku je nezbytné použít zařízení, které je schopné generovat pakety rychle, ale také na přesně nastavené rychlosti. Pokud není zdroj paketů dostatečně rychlý a přesný, pak není zřejmě možné zaručit smysluplnost výsledků. V našem případě pro generování paketů bylo zvoleno zařízení Spirent SPT-2000A.

V použitém Spirentu SPT-2000A jsou nainstalovány dve rozšiřující moduly. První modul obsahuje 2 SFP+ porty, které jsou schopny přenést až 10 Gb/s. Druhá karta obsahuje čtyři RJ45 porty a čtyři SFP sloty. Každý RJ45 port je spárován s jedním SFP a je tak možné používat jen jeden port z dané dvojice.

Základní vlastností Spirent SPT-2000A je generování paketových toků. Toky jsou definovány v tzv. streamblocku (dále jen SB). V každém SB je možné definovat hodnoty jednotlivých položek v hlavičkách podporovaných protokolů (pro měření jsou podstatné protokoly Ethernet, IP, TCP/UDP) a obsah na aplikační vrstvě. U některých vlastností je možné povolit vybírání hodnot z předvolené množiny a to buď deterministicky, nebo náhodně. Tyto množinové volby lze aplikovat na IP a MAC adresy, TCP/UDP porty a případně další, pro prováděné měření nepodstatné položky hlaviček. Množinové volby umožňují vygenerovat velké množství toků z jediného SB.

Na základě níže popsaných vlastností a problémů spojených s TCP komunikací je měření prováděno nad protokolem UDP.

Při konfiguraci SB je možné nastavit u TCP hlavičky příznaky pouze staticky, tedy pro všechny pakety ze SB stejně. Kvůli této vlastnosti není možné generovat pouze jedním SB validní TCP toky, které vyžadují pro první paket z toku mít nastavený příznak SYN [28]. Tento problém je možné obejít dvěma SB, kde první SB nejdříve vygeneruje pro každý tok jeden paket s nastaveným příznakem SYN. Následně se spustí druhý SB, který bude odesílat TCP pakety bez příznaku SYN.

V případě zapojení se Spirentem je uzel NAT možné zapojit tak, že je připojen na dvě rozhraní Spirentu. Jedno rozhraní pakety vysílá a druhé je zachycuje pro výpočet statistik. Při generování TCP komunikace je dle standardu TCP [28] vyžadováno, aby bylo dodrženo potvrzování přijatých paketů. To znamená, že z druhého rozhraní Spirentu, kde se počítají statistiky, je potřeba odeslat potvrzující pakety. Tato funkcionalita u používaného modelu Spirentu není implementována.

Pro měření je mimo jiné podstatné zaručit v jistý okamžik určitou rychlost odesílání paketů na uzel NAT, pokud není možné rychlost garantovat, pak po získání počtu upravených paketů za sekundu, nelze určit kolik paketů zpracováno nebylo. V rámci TCP komunikace existuje mechanismus, který při detekci zahlcení sítě sníží rychlost odesílání, jedná se o tzv. congestion control [7]. Toto chování by mohlo znehodnotit měření v případě, kdy je uzel NAT přehlcen a začíná zahazovat některé příchozí pakety. To vytváří situaci, kdy při zahození paketů je detekováno zahlcení linky, což vede ke snížení rychlost odesílání paketů a dochází k porušení podmínky garance rychlosti odesílání.

3.2 Popis měření

Pro změření propustnosti byla zvolena metoda, kdy se v aplikaci nastaví určitý počet pravidel,která definují jak se má překlad provádět a postupně se zvyšuje zátěž linky. Zároveň se zvyšováním zátěže byl na výstupu uzlu NAT měřen počet paketů, které prošli překladem síťových adres.

Další měřená veličina je round-trip time (RTT), která udává celkovou dobu mezi odesláním zprávy od jednoho uzlu k druhému a zpět [1]. RTT umožňuje určit trend, jak zvyšující se hustota síťového provozu ovlivňuje rychlost zpracování paketů na uzlu NAT.

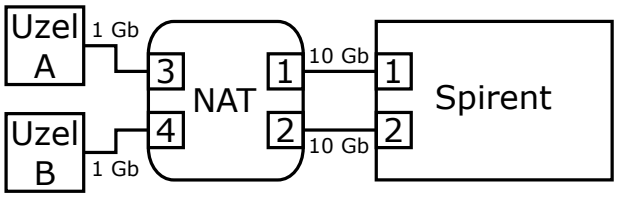
RTT bylo měřeno, tak že k uzlu NAT byly připojeny 2 uzly. Jeden byl umístěn do externí domény a druhý do lokální. Pro každou úroveň zátěže linky bylo pomocí nástroje traceroute provedeno měření RTT z lokální domény do externí. Měření bylo opakováno 30x s tím, že pro každé měření byla změněna adresa uzlu v lokální doméně, tak aby vždy došlo k vytvoření nového záznamu asociací.

Pro platformy s instrukční sadou x86 byl NAT měřen při nastavených 1 000, 3 000, 5 000 a 10 000 pravidel. V případě platforem ARM bylo měření provedeno pouze přehledově na 150 a 3 000 pravidlech a bez sledování RTT. Pořadí vyhodnocování pravidel bylo zvoleno, tak aby příchozí pakety musely být zkontrolovány na všechna pravidla a až poslední pravidlo vyhovělo.

Podle dostupných typů síťových karet byla zařízení měřena na postupně se zvyšující zátěži linky až po maximální možnou rychlost. U měření propustnosti NATu je podstatné zjistit, kolik paketů za sekundu (pps) je možné zpracovat. To znamená, že samotná velikost paketů není příliš důležitá (viz kapitola 3.3 Výsledky měření). Měření bylo provedeno na paketech o velikost 128 B.

Samotné měření probíhalo následovně:

1. Konfigurace pravidel překladu síťových adres.
2. Konfigurace Spirentu.
   (a) Výběr portů na Spirentu pro vstup a výstup paketů.
   (b) Nastavení správné cílové MAC adresy, tak aby na uzlu NAT pakety nebyly zahozeny.
   (c) Nastavení počátečního zatížení linky na 1 % maximální kapacity.
3. Konfigurace IP adresy uzlu A (lokální doména).
4. Konfigurace IP adresy uzlu B (externí doména).
5. Spuštění generovaní paketů na Spirentovi.
6. Měření RTT.
   (a) Změření RTT mezi uzly A a B.
   (b) Změna IP adresy na uzlu A.
   (c) Opakovaní měření RTT 30×.
7. Získání hodnoty počtu paketů za sekundu ze vstupního rozhraní Spirentu.
8. Zastavení generátoru paketů.
9. Zvýšení zatížení linky o 1 %.
10. Pokud je zatížení menší nebo rovno 100 % návrat na krok 5.

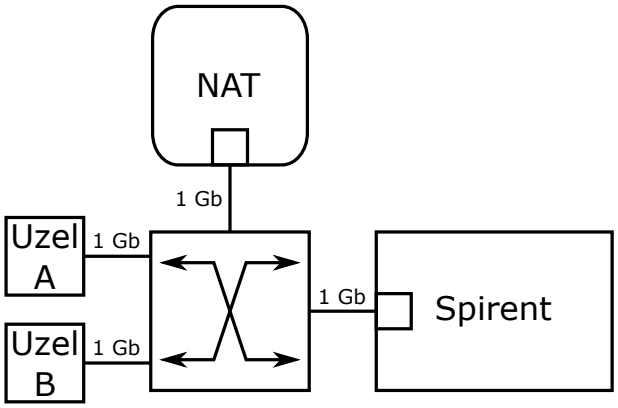


Obrázek 3.1: Topologie pro platformy s 10G porty

Pro měření byly použity 2 topologie. První topologie na obrázku 3.1, byla použita pro měření platforem s dvěma 10G porty a dvěma 1G porty. Porty spojené se Spirentem mají rychlost linky 10 Gbps a porty spojené s uzly mají rychlost 1 Gbps. Spirent generuje pakety na portu 1 a IP adresy generovaných paketů spadají do lokální domény uzlu NAT. Vygenerované pakety jsou doručeny na uzel NAT, kde je proveden překlad adres, a přeložené pakety jsou odeslány na druhý portu uzlu NAT, odkud jsou doručeny na druhý port Spirentu. Druhý port Spirentu provádí výpočet statistik přijatých paketů.

Uzly A a B slouží pro měření RTT. Uzel A má IP adresu z lokální domény, zatímco uzel B z domény externí. Uzel A se s každým zvýšením zátěže, kterou generuje Spirent, pokusí navázat spojení s uzlem B. Po nevázaní spojení získané RTT pro pozdější zpracování.

Topologie pro NAT s jedním 1G portem je na obrázku 3.2. Zde jsou všechna zařízení připojena ke gigabitovému přepínači. Spirent je připojen jen jedním portem a ten slouží jako port pro generování paketů, tak i jako rozhraní pro příjem paketů z uzlu NAT a výpočet statistik. Vytížení jedné linky oběma směry nezkreslí výsledky měření, protože linky s rychlostí jeden gigabit jsou plně duplexní [4].



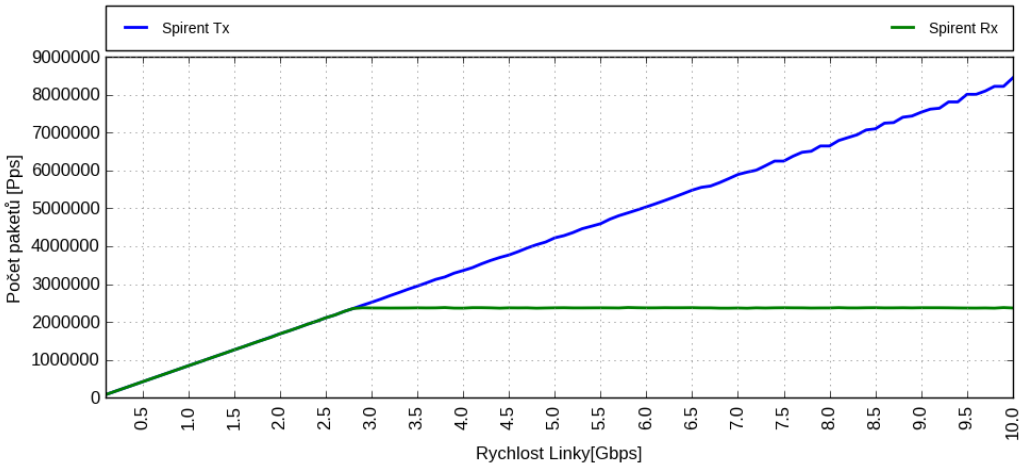
Obrázek 3.2: Obrázek 10 Topologie pro platformy s 1G porty

3.3 Výsledky měření

V následující části jsou rozebrány jen některé výsledky z provedených měření. Pokud není explicitně uvedeno jinak, všechna měření byla provedena na paketech o délce 128 B . Grafy všech naměřených výsledků je možné nalézt v příloze A.

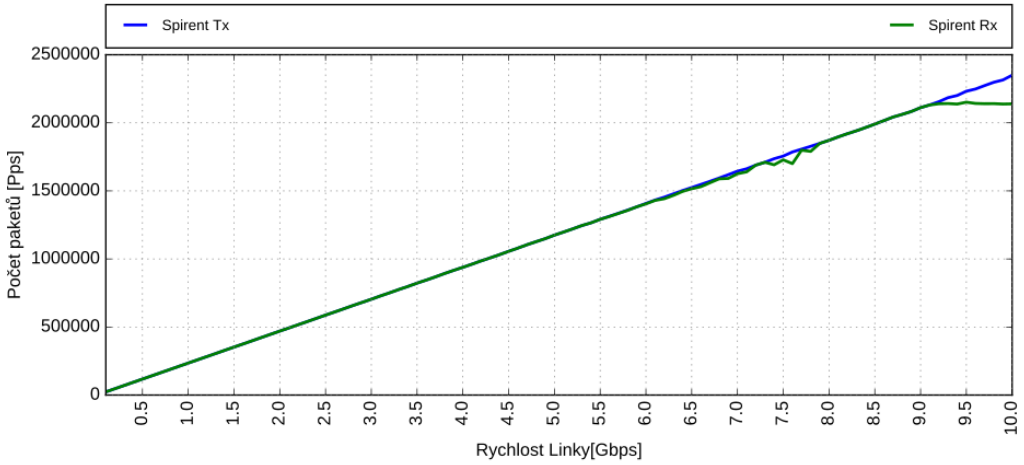
Na obrázku 3.3 je zobrazen graf propustnosti pro platformu s procesorem Intel Xeon E5-2620, při nastavených 1 000 pravidel. Na tomto grafu je možné si všimnout, že při narůstající

zátěži linky je počet zpracovaných paketů stejný jako počet přijatých až přibližně po 2,5 milionu paketů za sekundu. Tato hodnota, pak při dalším zvyšováním zátěže linky zůstává neměnná. Podobný jev se vyskytuje u všech měřených platforem s 10G porty (viz grafy v příloze A).



Obrázek 3.3: Intel Xeon – graf závislosti propustnosti na zátěži linky, 1 000 pravidel

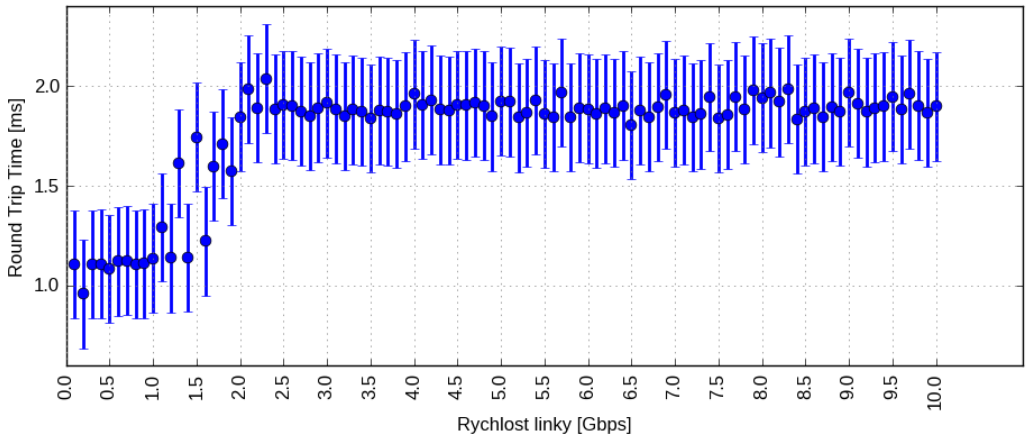
Stejné měření, jako je popsáno v minulém odstavci, bylo provedeno ještě jednou, ale s upravenou velikostí použitých paketů na 512 B. Výsledek je v grafu na obrázku 3.4. Jak lze vidět maximální propustnost z 2,8 Gbps vzrostla na 9,2 Gbps, ale počet zpracovaných paketů za sekundu je téměř shodný. Na základě tohoto zjištění jsou všechna ostatní měření prováděna pouze na paketech o délce 128 B.



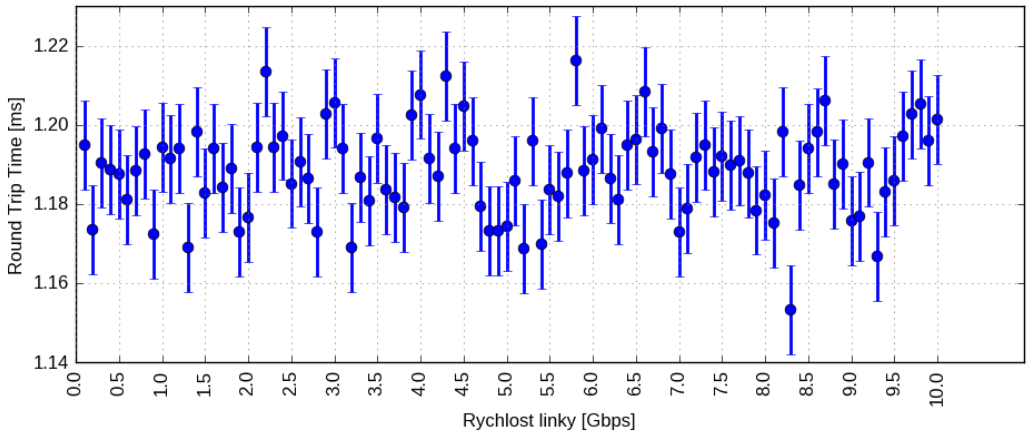
Obrázek 3.4: Intel Xeon – graf závislosti propustnosti na zátěži linky, 1 000 pravidel, 512B pakety

Graf na obrázku 3.5 zobrazuje závislost RTT na zatížení linky s konfigurací AMD Opteron 6376 a s 3 000 pravidly pro překlad adres. Na obrázku 3.6 je graf, který se od grafu 3.5 liší pouze v počtu použitých pravidel, konkrétně se jedná o 5 000. V případě NATu s 5 000 pravidly, jsou hodnoty RTT mnohem chaotičtější než v případě NATu s 3 000 pravidly. Ta-

kový markantní rozdíl při změně počtu pravidel se projevil pouze u platformy s procesorem AMD Opteron 6376. K tomuto chování dochází pravděpodobně kvůli tomu, že při zvýšení pravidel se více projeví režie koherence pamětí cache.

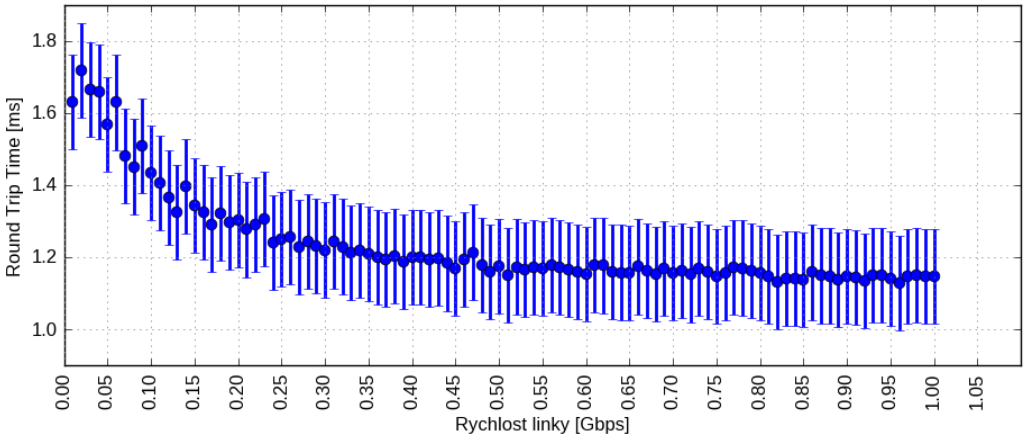


Obrázek 3.5: AMD Opteron – graf závislosti RTT na zátěži linky, 3 000 pravidel



Obrázek 3.6: AMD Opteron – graf závislosti RTT na zátěži linky, 5 000 pravidel

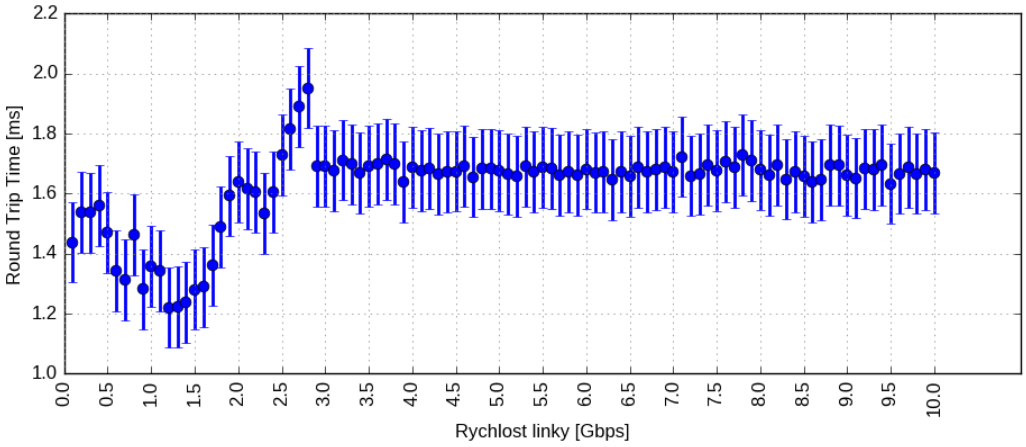
Na obrázku 3.7 je graf závislosti RTT na zatížení linky měřený při 10 000 pravidlech s procesorem Intel i3-6300. Jak je vidět s narůstající zátěží linky se RTT postupně snižuje. Toto chování způsobuje mechanismus, který se nazývá NAPI [16]. NAPI (New API) je aplikační rozhraní pro tvorbu ovladačů síťových rozhraní pro OS Linux. Standardně je při příchodu paketu na síťové rozhraní jádro OS upozorněno přerušením. Se zvyšujícím se počtem paketů na lince stoupá i počet přerušení. Zpracování přerušení trvá určitou dobu a v případě velkého počtu přerušení za krátkou dobu se může vyplatit, na místo čekání na přerušení a jeho následné zpracování, zahájit aktivní čekání na nové pakety. Při použití NAPI se při překročení meze počtu přerušení za určitou časovou jednotku automaticky přejde na aktivní čekání, což urychlí zpracování příchozích paketů.



Obrázek 3.7: Intel i3-6300 – graf závislosti RTT na zátěži linky, 10 000 pravidel

V grafu na obrázku 3.7 je možné velmi dobře pozorovat vliv NAPI. Při nízkém zatížení jsou pakety zpracovávány pomocí přerušení a doba, kdy se aktivně čeká je nízká nebo nulová. S tím jak narůstá zátěž linky, se doba aktivního čekání zvětšuje a některá měření RTT jsou zpracovány ihned na místo, aby pakety čekaly na zpracování přerušení.

NAPI se také projevuje u procesoru Intel Xeon E5-2620. Nejlépe to lze pozorovat při 10 000 pravidlech (obrázek 3.8). RTT nejdříve klesá, dokud vytížení linky nepřesáhne 1,5 Gbps. Následně, kvůli velkému počtu paketů, které je potřeba odbavit, se RTT začne opět zvětšovat. Přibližně při 3 Gbps (cca 2,5 milionu paketů za sekundu), což je maximální zatížení, které je na této platformě možné zpracovat bez ztrát paketů, se RTT stabilizuje.



Obrázek 3.8: Intel Xeon – graf závislosti RTT na zátěži linky, 10 000 pravidel

V tabulce 3.1 jsou výsledky měření zařízení s ARM procesory. Pro srovnání jsou také uvedeny hodnoty naměřené s procesorem Intel i3-6300. Jak lze výsledků vidět, maximální počet zpracovaných paketů za sekundu je s procesorem Intel i3-6300 až 25× vyšší, něž je tomu u zařízení s procesory ARM.

Zařízení	**NAT 150 [Kpps]**	**NAT 3000 [Kpps]**
Intel i3-6300	—	850
OLinuXino Lime A20	33	33
HardKernel Odroid-C1+	75	80

Tabulka 3.1: Porovnání ARM platforem s Intel i3-6300

Kapitola 4

Překlad síťových adres nad knihovnou DPDK

Pokud paket dorazí na běžný počítač, tak je zpracován jádrem OS. Běžné OS mají být univerzální a nejsou cíleny, pro použití pouze v oblasti zpracování paketů. Z tohoto důvodu je vhodné pro urychlení síťových aplikací, obejít jádro OS a pakety zpracovávat úplně ve vlastní režii. Existují knihovny, které usnadňují takovýto přístup ke zpracování paketů, jedná se například o Netmap[1], nebo DPDK[2]. Dále se budeme věnovat knihovně DPDK (Data Plane Development Kit), protože implementace překladu síťových adres, která je popsána dále v textu, je postavena právě nad touto knihovnou.

4.1 DPDK

DPDK knihovna je tvořena z několika klíčových částí. Zejména se jedná o EAL, poll-mode ovladače síťových karet, a z několika dalších knihoven. Podrobnější popis je v následujících kapitolách.

4.1.1 EAL

Komunikaci s operačním systémem a přístup k nízkoúrovňovým zdrojům zajišťuje tzv. EAL (Environment Abstraction Layer). Základní funkcionalita je následující [8]:

Správa procesů
: EAL umožňuje rozpoznat počet dostupných procesorů a jader. Na základě této informace je odvozena hodnota *lcores* (logical cores), který udává maximální počet vláken pro paralelní zpracování uvnitř DPDK aplikace. EAL dále zajišťuje samotné spuštění vláken a také nastavuje tzv. *affinitu*[3].

Abstrakce operační paměti
: DPDK aplikace se běžně spouští s podporou tzv. hugepages. Hugepages jsou speciální stránky paměti, které mají v Linuxu obvykle velikost 2 MB, ale mohou mít i více. Často se používají právě u aplikací vyžadující velké množství operační paměti [3]. Alternativně je možné na místo hugepages používat běžné stránky. EAL zajišťuje

[1] http://info.iet.unipi.it/ luigi/netmap/
[2] http://dpdk.org/
[3] Uzamčení vlákna na konkrétní jádro procesoru.

rezervaci paměti, se kterou se v rámci DPDK aplikace pracuje (tato paměť je poté dostupná pomocí dalších knihoven, které jsou součástí DPDK) a to bez ohledu na to, zda se na pozadí používají hugepages nebo běžné stránky.

Zpracování přerušení
Toto rozhraní umožňuje zaregistrovat některé funkce, pro zpracování přerušení, které mohou být vyvolána z některých vnitřních funkcí DPDK. Příkladem přerušení může být odpojení Ethernetového kabelu ze síťového adaptéru.

Správa časovačů
EAL umožňuje nastavit některé funkce, které mají být spuštěny až v určitý čas.

Identifikace CPU Protože DPDK aplikace může být přeložena s určitými požadavky na hardware (například dostupnost instrukcí SSE4.1), tak EAL umožňuje identifikovat na jaké hardwarové sestavě byla DPDK aplikace spuštěna. Po identifikaci hardwaru a zkontrolování, zda splňuje dané požadavky, je rozhodnuto, jestli DPDK aplikace bude spuštěna.

Jednou z výhod EAL, je zjednodušení práce s vlákny, kdy se programátor nemusí zabývat jejich spouštěním a zastavováním. Vlákna jsou na začátku spuštění DPDK aplikace vytvořena a dokud jim není přiřazena funkce, tak na přiřazení čekají. Inicializace programu a struktur probíhá ve vláknu, které se označuje jako *master*, zatímco ostatní vlákna jsou označena jako *slave*.

4.1.2 Ring

Součástí DPDK je knihovna obsahující implementaci fronty, jedná se o tzv. ring library. Základní charakteristika této fronty je následující [22]:

- Po inicializaci má danou pevnou velikost.
- Podporuje režimy:
  - Jeden konzument, jeden producent.
  - Více konzumentů, více producentů.
  - Kombinace předchozího.
- Implementace je bezzámková. (Pro zabránění chyby typu race condition, jsou použity atomické operace.)

Tato fronta je určena pro komunikaci mezi jednotlivými procesy DPDK aplikace. Jedno z možných použití může být takové, že po spuštění DPDK aplikace je spuštěno n vláken a vytvořeno $n-1$ front. Jedno vlákno přijímá veškeré pakety ze síťového adaptéru a na základě některého z dostupných algoritmů posílá jednotlivé doručené pakety do jedné z $n-1$ front. Cílem takovéto implementace je obvykle paralelizovat zpracování jednotlivých paketů a zvýšit tím propustnost celé aplikace.

4.1.3 Mempool

Jak bylo řečeno EAL zajišťuje inicializaci paměti, která může být tvořena hugepages, nebo standardními stránkami. Pro přístup k této paměti je zapotřebí použít API, které je součástí

knihovny rte_memzone. Knihovna rte_memzone umožňuje přistupovat k paměti, kterou na začátku spuštění DPDK aplikace alokoval EAL. I přesto, že funkce z knihovny memzone lze použít přímo v DPDK aplikaci, jeho použití je často skryto uvnitř dalších knihoven DPDK, které umožňují pracovat s pamětí efektivněji a na vyšší úrovni abstrakce. Knihovny, které v DPDK pracují s pamětí, pro svoji alokaci používají funkce z knihovny memzone, například se jedná o ring library, nebo o mempool library.

Mempool (zkratka z angl. memory pool) slouží pro rychlou alokaci datových struktur o předem známé velikosti a typu. Při inicializaci mempoolu je nastaven počet a velikost objektů, které bude možné z daného mempoolu získat (do mempoolu lze nepoužívané objekty zase vrátit). Další vlastností mempoolu je možnost nastavit, jak budou jednotlivé alokovatelné objekty zarovnány v paměti. Vhodné nastavení zarovnání objektů může vést k lepšímu využití kanálů u DRAM pamětí, a tím k vyšší propustnosti [15]. Mempool knihovna implementuje a zprostředkovává API k používání mempool objektů v DPDK aplikacích.

4.1.4 Message buffer

DPDK je knihovna pro zpracování síťového provozu, což znamená, že DPDK aplikace musí zpracovávat síťové rámce. Datová struktura, která se v DPDK pro rámce používá se nazývá *message buffer*. Message buffer není vytvořen výhradně pro ukládání rámců v paměti, pokud je to vhodné, pak jej lze použít i pro jiná data [14].

Základní charakteristiky objektu message bufferu:

Řtězení message bufferů
: Každý message buffer má danou maximální velikost, která se udává v bajtech (ve výchozím nastavení DPDK verze 2.2.0 je velikost nastavena na 2 KB). Maximální velikost message bufferu je definována za překladu knihovny DPDK. Pokud je třeba pracovat s delšími rámci než je stanovená horní hranice, lze využít mechanismus řetězení Message Bufferů. Řetězení se používá např. při zpracování Jumbo rámců.

Headroom
: Message buffer může obsahovat tzv. headroom. Headroom obecně slouží pro předřazování hlaviček (např. IP, TCP, UDP, ICMP...) před aktuální obsah message bufferu. Pokud se v DPDK aplikaci headroom k takovýmto účelům nepoužívá, pak je jej možné použít pro uložení metadat, která náleží k obsahu uloženému v message bufferu[4].

4.1.5 Hashovací tabulka

DPDK obsahuje implementaci hashovací tabulky. V rámci API je při vytváření hashovací tabulky možné nastavit [11]:

- Maximální počet záznamů v hashovací tabulce.
- Velikost klíče v bajtech.
- Hash funkci pro převod klíče na index.

U vícevláknových aplikací je možné k hashovací tabulce přistupovat současně pouze v případě, kdy se v ní vyhledává požadovaný záznam. Pokud je vyžadováno provádění úprav, pak je potřeba, aby vlákna byla synchronizována a nikdy nenastala situace, kdy dvě vlákna upravují obsah jedné hashovací tabulky. Zejména se jedná o operace přidávání a mazaní.

[4] Takovéto použití není v DPDK dokumentováno, ale pouze se využívá jinak nepoužité místo v *headroomu*

4.1.6 Ovladače síťových karet

Pro zpracování paketů mimo jádro operačního systému je vyžadováno, aby pakety z vybraných síťových rozhraních byly doručeny přímo DPDK aplikaci. Z tohoto důvodu je součástí DPDK specifikace API pro tvorbu ovladačů síťových adaptérů [18]. Zároveň jsou k DPDK přibaleny ovladače pro vybrané síťové adaptéry.

Před spuštěním samotné DPDK aplikace se k jednotlivým síťovým rozhraním adaptérů, které se mají používat v rámci DPDK aplikace, přiřadí jeden z DPDK ovladačů. Po přiřazení DPDK ovladačů k síťovým rozhraním není možné tato rozhraní nadále používat přímo z operačního systému, ale jsou dostupná pouze DPDK aplikaci.

Při tvorbě ovladačů pro DPDK se očekává, že jejich implementace nepoužívá při příchodu nového rámce přerušení pro vyvolání obslužné rutiny, ale na rámce se čeká v aktivní smyčce. Tento přístup je efektivní z pohledu nízké latence, při doručení paketu. Nevýhoda spočívá v množství procesorového času, kdy není prováděna žádná užitečná práce a je tak spotřebovávána zbytečná elektrická energie.

Vzhledem k tomu, že se v DPDK používají nesystémové ovladače, je možné vytvořit i takové, které ke své funkci nepotřebují reálný síťový adaptér. Například je možné použít rozhraní AF_INET v OS Linux, nebo pro vstup a výstup použít PCAP soubory. Takováto rozhraní jsou hlavně vhodná pro ladění a testování.

4.1.7 KNI

KNI (zkratka z angl. Kernel NIC Interface) umožňuje aplikaci DPDK vytvořit virtuální síťová rozhraní v OS Linux. Z pohledu operačního systému se jedná o plnohodnotná rozhraní, která mají svoji MAC adresu, mohou mít IPv4 i IPv6 adresu atp. Libovolný paket, který je operačním systémem z tohoto rozhraní přečten, nebo je do něj naopak zaslán, prochází všemi fázemi, které jsou v Linuxu nad paketem provedeny. Například je tak možné nad tímto rozhraním nastavit firewall pomocí iptables.

Z pohledu aplikace DPDK se jedná o dvě FIFO fronty. Jedna fronta slouží pro zápis paketů, které jsou následně doručeny operačnímu systému. Druhá fronta slouží pro čtení, což jsou pakety, které operační systém vyslal daným virtuálním rozhraním.

Tato funkcionalita se na první pohled může zdát zbytečná, když celým účelem DPDK aplikace nezpracovávat pakety v jádře operačního systému. Uvažujme následující scénář. Předpokládejme, že existuje DPDK aplikace, která provádí standardní L3 směrování se dvěma síťovými porty. Každý port tak má svoji IP adresu. Dokud chodí pakety, které jedním portem vstupují a druhým vystupují, pak aplikace řádně funguje. Problém nastává ve chvíli, kdy dorazí paket, jehož cílová IP adresa je stejná, jako adresa rozhraní přiřazeného k DPDK aplikaci. Například se může jednat o správce sítě, který se snaží připojit k SSH serveru na počítač, kde běží tato DPDK aplikace. Jak bylo řečeno v minulé kapitole, tak rozhraní přiřazená DPDK aplikaci nejsou dostupná operačnímu systému a tak máme na výběr dvě možnosti:

1. Paket můžeme zahodit s tím, že daný protokol nepodporujeme, alternativně je možné implementovat vlastní TCP/IP zásobník, SSH server a případně další desítky služeb, které chceme podporovat.

2. Využít KNI, kdy paket odešleme operačnímu systému, kde může běžet běžný SSH server.

4.2 Překlad síťových adres s DPDK

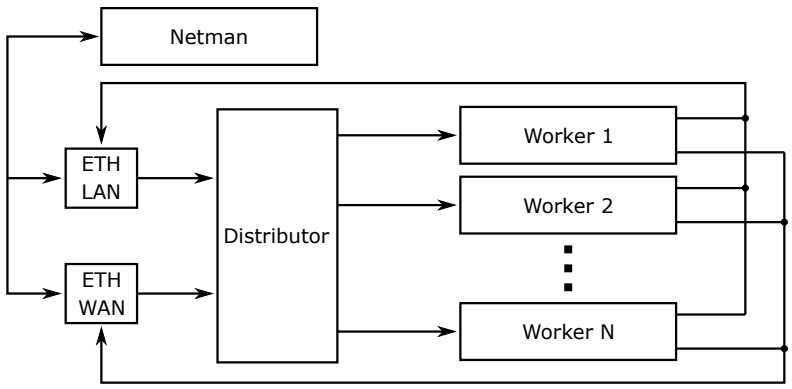
V době vypracování této diplomové práce vznikla nová implementace překladu síťových adres (dále v textu označována jako NAT DPDK). Cílem této práce je dostupnou implementaci překladu síťových adres rozšířit o překlad adres na 7. vrstvě ISO/OSI modelu, obecně označováno jako *ALG* (zkratka z angl. Application-level gateway).

4.2.1 Základní architektura aplikace NAT DPDK

NAT DPDK se skládá ze 3. základních částí (viz obrázek 4.1). První částí je *distributoru*. Jeho hlavní úkol je zprostředkovat rozdělení příchozích paketů z Ethernetových rozhraní na jednotlivá jádra, kde jsou dále zpracovávány paralelně. *Distributorem* se podrobněji zabývá kapitola 4.2.2. Na jednotlivých jádrech běží tzv. *worker*. *Worker* získá od distributoru paket, provede překlad adres a upravený paket odešle přes síťové rozhraní. Problematikou *workerů* se zabývá kapitola 4.2.4.

V Ethernetových sítí je při odesílání rámců mezi jednotlivými uzly potřeba vyplnit Ethernetovou hlavičku. To znamená, že pokud chceme odeslat rámec do sítě, zpravidla známe všechny položky Ethernetové hlavičky s výjimkou cílové MAC adresy. V IP sítích ze směrovacích tabulek známe IP adresu následujícího uzlu, kam se paket má odeslat. K získání MAC adresy slouží protokol ARP.

NAT DPDK pro získání cílových MAC adresy používá *netman* objekt. *Netman* využívá nástroj *arping*, který prostřednictvím ARP protokolu získá z počítače na dané IP adrese vyžadovanou MAC adresu a vypíše ji na standardní výstup. *Netman* převezme tuto informaci a uloží ji do vnitřních struktur, odkud je pak načítána *workery* při vyplňování Ethernetových hlaviček.



Obrázek 4.1: NAT DPDK - Architektura

4.2.2 Distributor

Hlavním úkolem *distributoru* je rozdělit příchozí pakety na jednotlivá jádra. Na obrázku 4.2 je znázorněno blokové schéma distributoru.

Pakety přicházejí na Ethernetová rozhraní, která jsou přidělena DPDK aplikaci, odkud je síťový adaptér vkládá do fronty. První krok distributoru je získání paketů z paketové

fronty, tato operace se nazývá *deque*. Pro snížení počtu paměťových transakcí, je naráz vyčtena dávka paketů (ve výchozím nastavení se jedná maximálně o 32 paketů). Při provedení deque jsou pakety uloženy v message bufferu.

Z každého message bufferu je provedena extrakce a zpracování hlaviček síťových protokolů, zejména se jedná o Ethernetovou hlavičku, IP hlavičku, TCP/UDP/ICMP hlavičku. Zpracované informace jsou následně uloženy do headroom uvnitř message bufferu. Od této chvíle jsou zpracované hlavičky dostupné kterékoliv následující procesní části.

Po zpracování síťových hlaviček je provedeno rozhodnutí, zda se má celý paket zpracovat v NAT DPDK, nebo má být poslán přes KNI do jádra operačního systému, či má být rovnou předán na výstupní rozhraní. Toto rozhodnutí je provedeno na základě následujících kritérií:

- Pokud je cílová IP adresa paketu shodná s IP adresou rozhraní, na které dorazil, pak je poslán do OS.
- Pokud je na třetí vrstvě ISO/OSI modelu jiný protokol, než IPv4 nebo IPv6, pak je odeslán do OS.
- Pokud je na čtvrté vrstvě ISO/OSI modelu jiný protokol, než TCP, UDP nebo ICMP, pak je odeslán do OS.
- Pokud je na třetí vrstvě ISO/OSI modelu IPv6, pak je odeslán rovnou na výstupní rozhraní (neprovádí se překlad adres).
- V ostatních případech je zpracován v NAT DPDK.

Předtím než je paket předán workerovi, musí být určeno, která instance workeru daný paket získá. To je důležité, protože každé jádro má vlastní asociační tabulku. Nevhodně zvolené jádro by mohlo provést novou asociaci adres, i když jiná asociace pro danou IP adresu existuje. Podrobnější popis o asociačních tabulkách je v následující kapitole.

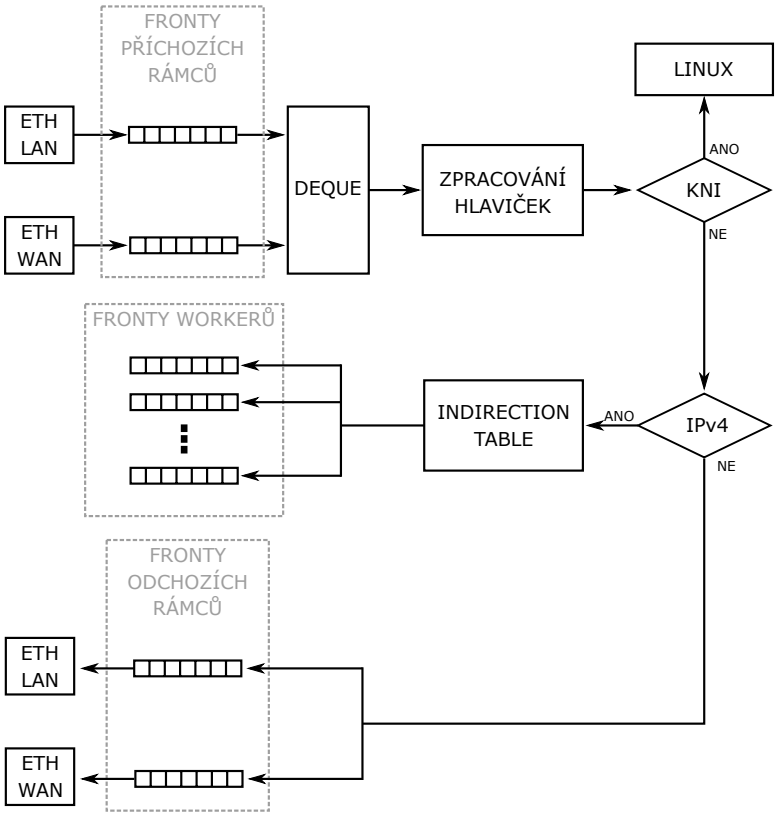
Rozhodnutí na které jádro bude paket předán se provádí pomocí tzv. *indirection table*. Jedná se o hashovací tabulku, ve které je počet záznamů roven počtu vláken s workery. Pokud je zpracovávaný paket původně z LAN sítě (lokální domény), pak je spočítána hash z cílové IP adresy. Jestliže doručený paket pochází z WAN sítě (externí domény), pak je spočítána hash ze zdrojové IP adresy. Vypočítaná hash slouží jako klíč do *indirection table*, ve které je k dané položce přidělen worker. Zjevně tento způsob výpočtu zajistí přidělení paketů náležících do jednoho relačního toku na jedno vlákno.

Posledním krokem, který distributor vykoná, je zařazení paketu do jedné ze dvou front, které worker má. Fronty jsou pojmenovány *outbound* a *inbound*. Distributor na základě původu paketu určí která z těchto dvou front je pro daný paket správná. Outbound fronta je určena pro pakety, které jsou původem z lokální domény (byly vyčteny z Ethernetového rozhraní připojené k LAN). Inbound fronta naopak slouží pro pakety z externí domény (WAN rozhraní).

4.2.3 Implementace asociační tabulky v NAT DPDK

Pro pochopení proč a jaké úkony musí worker vykonat, je vhodné porozumět implementaci asociační tabulky v NAT DPDK. Asociační tabulky jsou implementovány formou hashovacích tabulek.

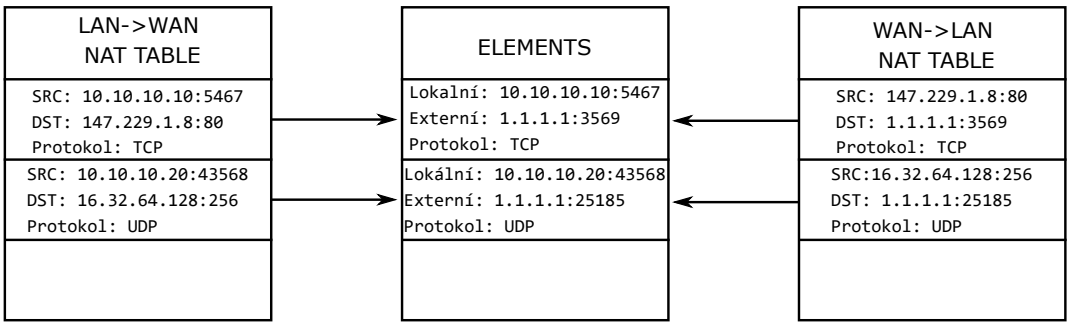
Klíčem do asociačních tabulek je tok, což je pětice (zdrojová adresa, zdrojový port, cílová addresa, cílový port, L4 protokol). Je zřejmé, že klíč vytvořený z hlaviček paketu, který přišel



Obrázek 4.2: NAT DPDK - Distributor

z lokální domény bude jiný, než klíč z paketu, který přišel z externí domény. Na základě této skutečnosti jsou pro každý worker vytvořeny dvě tabulky, které jsou označovány jako *LAN → WAN NAT table* a *WAN → LAN NAT table*. Obě tabulky obsahují ukazatel do tabulky *elements*, ve které jsou uloženy informace potřebně pro překlad adres. Nejdůležitějšími položkami v tabulce elements jsou lokální a externí IP adresa, lokální a externí L4 adresa a použitý L4 protokol.

Na obrázku 4.3 je příklad asociační tabulky NAT DPDK, kde jsou vytvořeny asociace pro dvě spojení. První je mezi lokální adresou 10.10.10.10:5467 a externí adresou 147.229.1.8:80, použitý komunikační L4 protokol je TCP. Obdobně je vytvořeno druhé spojení. V *LAN → WAN NAT table* a *WAN → LAN NAT table* je uveden klíč, který zpřístupňuje ukazatel do tabulky *elements*. V tabulce *elements* jsou pak základní informace potřebné pro překlad.



Obrázek 4.3: NAT DPDK - Asociační tabulka

Z asociační tabulky jsou odstraňovány záznamy tak, aby nedošlo k úplnému vyčerpání volných záznamů. Pro identifikaci, které záznamy mohou být uvolněny, elements obsahují položku *timestamp*, která udává poslední přístup k danému záznamu. U TCP protokolu je navíc sledován stav, ve kterém se spojení nachází a pokud je detekováno ukončení spojení (pomocí příznaků FIN), tak je záznam z tabulek vymazán. Pro možnost vytváření statistik je v záznamu tabulky *elements* uložen počet paketů a bajtů, kolik bylo v daném toku zpracováno.

NAT DPDK obsahuje ještě další hashovací tabulky velmi podobné těm asociačním. Jedná se o tzv *1:1 NAT table*. Tyto tabulky ukládají mapování jedné konkrétní lokální adresy na jednu konkrétní externí adresu. Toto mapování je součástí konfigurace NAT DPDK. Stejně jako u asociační tabulky existují dvě verze 1:1 NAT tabulky, konkrétně $LAN \rightarrow WAN$ a $WAN \rightarrow LAN$. $LAN \rightarrow WAN$ tabulka je použita pro pakety, které přišli z lokální domény. Jejím klíčem je zdrojová IP adresa a hodnotou, kterou obsahuje, je přidělená adresa z externí domény. Klíčem do $WAN \rightarrow LAN$ 1:1 tabulky je cílová IP adresa z paketů, které pochází z externí domény. Hodnoty uložené v $WAN \rightarrow LAN$ tabulce jsou přidělená lokální IP adresa a informace, zda překlad je restriktivní, či není (viz kapitola 2.1.1). Tyto tabulky jsou vytvořeny po spuštění aplikace a nadále nejsou modifikovány.

Poslední významnou tabulkou je *DNAT table*. Opět se jedná o hashovací tabulku a tentokrát je pouze jediná. DNAT tabulka je stejně jako tabulky 1:1 vytvořena při spuštění programu a od té doby se zůstává stejná. Tabulka obsahuje mapování trojice (IP adresa, port, L4 protokol) z externí domény na trojici z lokální domény s tím, že se neurčuje, který uzel z externí domény bude moci navázat spojení. Tato technika se také nazývá *port forwarding*. Je zřejmé, že je tato tabulka použita pouze pro pakety, které přicházejí z externí domény.

4.2.4 Worker

Hlavním cílem workeru je provést překlad síťových adres pro paket, který vyčte z jedné ze svých front. Způsob zpracování každého paketu je určen tím, z jaké fronty je paket vyčten. Pokud se získá z outbound fronty, pak je proveden překlad lokální adresy na adresu externí. Jestliže je vyčten z inbound fronty, provede se překlad v opačném směru. Nejdříve si popíšeme některé operace, které worker provádí. Následně bude uveden postup, v jakém pořadí jsou tyto operace prováděny na základě toho, z jaké fronty pochází.

Main update
Jedná se o operaci, která provádí aktualizaci statistik (počet přijatých/odeslaných paketů a bajtů) v tabulce *elements*.

1:1 update
Pokud zpracovávaný paket má záznam v 1:1 NAT tabulce, pak tato operace zkonvertuje tento záznam spolu s hlavičkami zpracovávaného paketu a vytvoří novou asociaci do tabulky *elements*. Následně přidá pro adekvátní klíče do *LAN → WAN/WAN → LAN NAT tabulky* ukazatele na nový záznam v *elements*.

DNAT update
Stejně jako *1:1 update* přidává asociace do tabulky *elements*, ale pouze pro pakety, které mají záznam v *DNAT tabulce*.

IP a port allocate
Jedná se o dvě operace, které dohromady ze seznamu volných asociačních adres získají trojici (IP adresa, L4 adresa, L4 protokol). Tato trojice v době získání není nikde v překladu adres používána, současně je také z rozsahu adres externí domény. Získaná trojice je v NAT DPDK zarezervována a dokud tento stav trvá, nemůže být znovu přidělena.

New update
Operace vytvoří z hlaviček příchozího paketu a rezervované trojice (viz *IP a port allocator*) novou asociaci a stejně jako *1:1 update*, i *new update* přidá záznam do tabulky *elements* a nastaví ukazatele v *LAN → WAN/WAN → LAN NAT tabulkách*.

Mangling
Tato operace na základě nalezené/vytvořené asociace upraví L3 a L4 adresy v paketu. Pokud paket dorazil z lokální domény, pak jsou přepsány zdrojové adresy. V opačném případě jsou přepsány cílové adresy.

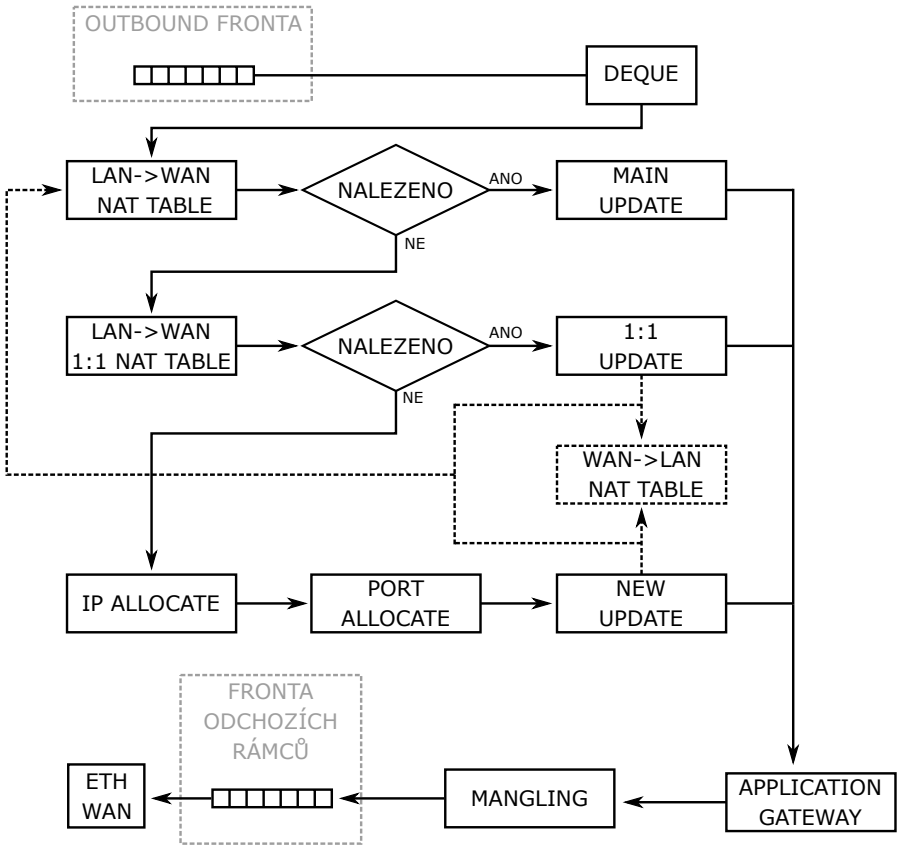
Drop
Tato operace označuje zastavení překladu daného paketu. V jejím rámci se provede uvolnění přiděleného message bufferu pro daný paket a paket se do statistik započítá jako zahozený.

Worker provádí zpracování paketů v jeho hlavní smyčce. Nejdříve se zpracují pakety z *outbound* fronty. Následně zpracuje pakety z *inbound* fronty. V jednom cyklu je možné z každé fronty odbavit pouze určitý maximální počet paketů, počet je udán konfigurací NAT DPDK. Toto omezení zamezuje problému vyhladovění front, pro případ, kdyby na jednu frontu přicházely pakety rychleji, než by byly zpracovávány.

Při spuštění NAT DPDK je nakonfigurováno, jak dlouho mohou být uloženy záznamy v tabulce *elements*, aniž by k nim bylo přistoupeno kvůli překladu. Worker pro uvolnění těchto záznamů, po zpracování paketů z front, projde N položek z *elements*, kde N je nastaveno při spuštění NAT DPDK. Pro každou položku, kterou projde vyhodnotí, zda nemá být uvolněna. V následujícím cyklu hlavní smyčky workeru je zpracováno dalších N záznamů od pozice, kde v předchozím průchodu skončil.

Postup zpracování paketu z outbound fronty (viz obrázek 4.4):

1. Paket je vyčten z fronty (operace *deque*).
2. Pro paket se provede vyhledání asociace v *LAN → WAN NAT tabulce*.
3. Pokud záznam není nalezen pokračuje se krokem 4, v opačném případě se provede následující:
   (a) Pro daný paket se spustí *main update* (aktualizace statistik pro daný tok).
   (b) Paket je předán na zpracování ALG algoritmy (krok 8).
4. Provede se vyhledání v *LAN → WAN 1:1 NAT tabulce*.
5. Pokud pro daný paket není nalezen záznam pokračuje se krokem 6, v opačném případě:
   (a) Pro daný paket a nalezený záznam je provedena operace *1:1 update*.
   (b) Paket je předán na zpracování ALG algoritmy (krok 8).
6. Pro tok, do kterého paket náleží, nebyla nalezena žádná asociace. Z tohoto důvodu je toku přidělena nová asociace, která je v externí doméně volná (operace *IP a port allocate*).
7. S paketem a novou asociací se provede operace *new update*.
8. Zkontroluje se zda je potřeba provést překlad i na úrovni L7, pokud ano pak se provede.
9. Nahrazení L3 a L4 adres v hlavičkách paketu (operace *mangling*)
10. Paket se vloží do fronty na síťovém adaptéru, odkud bude vyslán do externí domény.

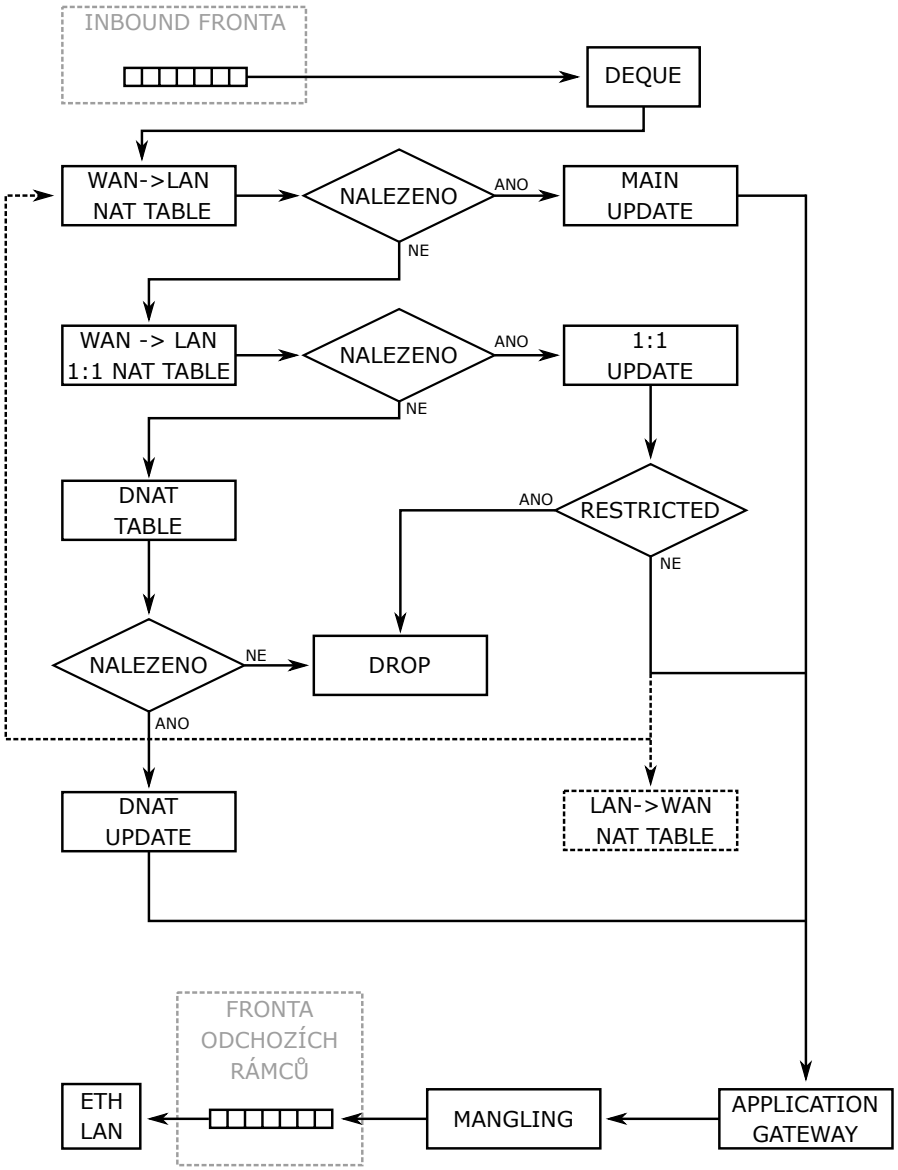


Obrázek 4.4: NAT DPDK - Zpracování odchozích paketů

Postup zpracování paketu z inbound fronty (viz obrázek 4.5):

1. Paket je vyčten z fronty (operace *deque*).
2. Provede se prohledání asociací v hlavní tabulce WAN → LAN.
3. Pokud záznam není nalezen pokračuje se krokem 4, v opačném případě se provede následující:
   (a) Pro daný paket se spustí *main update* (aktualizace statistik pro daný tok).
   (b) Paket je předán na zpracování ALG algoritmy (krok 8).
4. Provede se vyhledání v *WAN → LAN 1:1 NAT tabulce*.
5. Pokud pro daný paket není nalezen záznam pokračuje se krokem 6, v opačném případě:
   (a) Provede se kontrola na typ nalezeného záznamu. Pokud je záznam označen jako restriktivní, pak je další zpracovávání paketu zastaveno (operacce *drop*).
   (b) Pro daný paket je spuštěn *1:1 update*.
   (c) Paket je předán na zpracování ALG algoritmy (krok 8).
6. Provede se vyhledání v *DNAT tabulce*.

7. Jestliže záznam není nalezen, pak je další zpracovávání paketu zastaveno (operace *drop*).
8. S paketem je proveden *DNAT update*.
9. Zkontroluje se zda je potřeba provést překlad i na úrovni L7, pokud ano pak se provede.
10. Nahrazení L3 a L4 adres v hlavičkách paketu (operace *mangling*)
11. Paket se vloží do fronty na síťovém adaptéru, odkud bude vyslán do lokální domény.



Obrázek 4.5: Zpracování příchozích paketů

Předchozí popis zpracování paketů je určen výhradně pro pakety, které na transportní vrstvě ISO/OSI modelu používají TCP/UDP protokol. Součástí NAT DPDK je implementace zpracovávající ICMP zprávy. Tato funkcionalita však není pro tuto práci podstatná a tedy nebude popsána.

Kapitola 5

Zpracování aplikačních protokolů v NAT DPDK

Pokud vývojáři síťových aplikací a protokolů chtějí, aby jejich implementace byla implicitně podporována překladem síťových adres, tak by se měli držet souboru doporučení uvedených v RFC 3235 [24]. Jedno z doporučení radí vyhnout se používání adresování (např. IP adres) v aplikačních datech. Existují situace, kdy se tohoto doporučení nelze držet, například je to běžné u signalizačních protokolů, které si potřebují předat informace nutné pro vytvoření nového spojení. Pokud je při překladu adres vyžadována podpora těchto protokolů, pak implementace překladu musí být schopna daný protokol rozpoznat a korektně upravit aplikační data. Tato technika se běžně v překladu adres označuje jako Application-level Gateway (zkráceně ALG).

Také existují protokoly, které nelze v sítích s překladem adres používat. Převážně se jedná o protokoly, které nějakým způsobem zamezují s manipulací adres. Například IPsec [12] je navržen přímo tak, aby detekoval změny v IP hlavičce. Pokud je při překladu změněna IP adresa, pak na cílovém uzlu je změna detekován a paket je zahozen. Další skupinou jsou protokoly, které překladu adres zabraňují šifrováním aplikačních dat[1].

Cílem této práce je rozšířit NAT DPDK o podporu vybraných aplikačních protokolů. Konkrétně se jedná o FTP (File Transfer Protocol), SIP (Session Initiation Protocol) a IRC DCC (Internet Relay Chat Direct Client-to-Client).

5.1 Rozhraní pro zpracování aplikačních protokolů

Na základě zdrojového/cílového TCP/UDP portu se u každého paketu, který je zpracován v NAT DPDK rozhoduje, zda paket má být upraven ALG algoritmy. Zdrojový port je použit pro pakety, které *worker* vyčetl z *inbound* fronty, zatímco cílový port je použit pro pakety z *outbound* fronty. Tento způsob rozhodování je založen na tom, že u paketů původem z lokální domény nelze předpovědět, jaké zdrojové porty budou při vytváření komunikace použity. Pro konkrétní protokoly lze určit, jaký bude cílový port použitý v paketech. To vychází z předpokladu, že daný protokol má registrované číslo portu u organizace ICANN (jedná se o tzv. *well-known* a *registered* porty). Samozřejmě je možné zpracovávat i pakety, které nepoužívají standardní porty.

V NAT DPDK jsou dvě tabulky pojmenované *Port map*. Každá tato tabulka má 65 535 položek. Jedna tabulka je určena pro pakety používající TCP protokol a druhá pro pakety

[1] Toto se týká pouze těch, které v aplikačních datech přenášejí adresy.

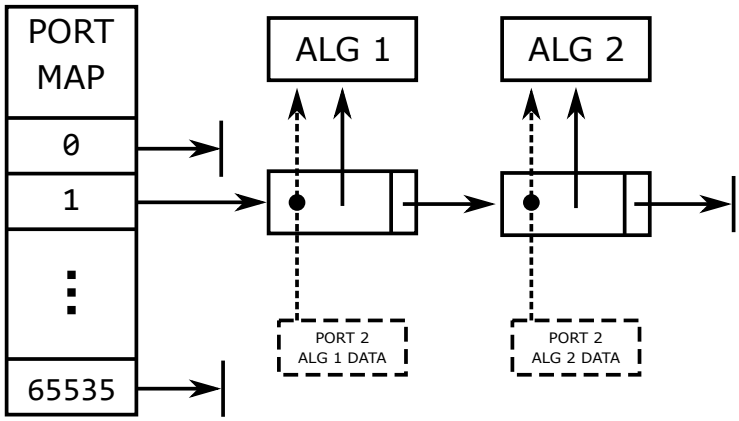
s UDP protokolem. Každá položka v tabulce *Port map* je ukazatel na zřetězený seznam a každý prvek v seznamu obsahuje tři ukazatele.

Ukazatel na ALG
Ukazatel na funkci, která s daným paketem provede úpravu aplikačních dat. Funkci se v parametrech předává:

- Ukazatel na message buffer, který obsahuje upravovaný paket.
- Typ fronty odkud worker paket vyčetl (inbound/outbound).
- Ukazatel na záznam do tabulky *Elements*, aby bylo možné určit na jakou adresu se má překlad provést.
- Ukazatel na uživatelská data (viz následující bod).

Ukazatel na uživatelská data
Při vytváření položky v seznamu je do ní možné uložit ukazatel na data, která budou dostupná při provádění překladu aplikačních dat. Tento ukazatel může být pro každý záznam jedinečný nebo u všech stejný. V případě, kdy je stejný pro více záznamů, pak je nutné dávat pozor, aby bylo při přístupu z více vláken zabráněno *race condition*.

Ukazatel na uvolňovací funkci
Uvolňovací funkce je zavolána ve chvíli, kdy je daný záznam odstraňován ze seznamu, aby mohli být uvolněni alokované zdroje.



Obrázek 5.1: Architektura rozhraní pro ALG

Na obrázku 5.1 je znázorněn příklad *Port map* tabulky. K portu jedna jsou přiřazeny dvě funkce (ALG 1 a ALG 2) provádějící překlad adres na aplikační vrstvě. Při zpracování paketu, který byl vyčten z *outbound* fronty, je jeho číslo cílového portu použito jako index do tabulky *Port map*. Pokud záznam obsahuje platný ukazatel, pak se projde celý seznam a z každého záznamu je spuštěna funkce pro úpravu aplikačních dat.

Každá implementace zpracování protokolu pro ALG v NAT DPDK tvoří modul. Například implementace pro protokol FTP tvoří jeden modul. Při překladu NAT DPDK je

možné zvolit, zda moduly budou součástí přeložené aplikace, nebo se bude jednat o dynamické knihovny.

Pokud je modul ve formě dynamické knihovny, pak se jeho načítaní provádí funkcí `load_alg_module`. Této funkci se parametrem předá název modulu, který má být načten a následně je zajištěno načtení symbolů z knihovny. V dalším kroku se z načteného modulu zavolá funkce `register_module_#nazev_modulu#` (například pro modul FTP se zavolá funkce `register_module_ftp`). V případě, že modul není ve formě dynamické knihovny, ale je součástí aplikace, pak je nutné zajistit zavolání registrační funkce v inicializaci programu.

Registrační funkce provádí inicializaci modulu. Jejím hlavním cílem je zavolat funkci `register_port`, která přidá záznam do tabulky *Port map*. Od této chvíle je možné pro pakety, které vyhovují zaregistrovaným portům, provádět překlad adres v aplikační vrstvě.

Obě tabulky *Port map* jsou sdíleny mezi *workery*. To znamená, že se do nich přistupuje paralelně. Pokud jde pouze o čtení dat z tabulky, pak paralelní přístup je možný. Ve chvíli, kdy se obsah tabulky nebo některý ze zřetězených seznamů začne měnit, pak se musí zajistit, aby přístup byl synchronizován, protože operace přidaní a odebrání nejsou atomické.

Nejjednodušší řešení je použít zámek, který umožní exkluzivní přístup do tabulky. To by znamenalo zamknutí a odemknutí zámku pro každý paket, což je neefektivní a hlavně to neumožňuje přístup více vláken v jednu chvíli, i když se provádí pouze čtení.

Druhou možností je použít typ zámku, který umožňuje přístup více vláknům naráz za podmínky, že nebudou *Port map* upravovat. Pokud o přístup do kritické sekce zažádá vlákno s úmyslem *Port map* upravit, pak zámek získá ve chvíli, kdy veškerá vlákna, které pouze čtou, sekci uvolní.

Předpokládejme, že použijeme zámkem, který povoluje více přístupů do kritické sekce, tak se stále pro každý paket musí provést jedno získání zámku a jedno navrácení. Toho se lze vyhnout tak, že využijeme vlastností instrukční sady x86_64.

Za předpokladu, že máme vhodně zarovnaná data, pak instrukce `MOV`, která provádí čtení (zápis) z (do) paměti, je atomická [13]. Toho je možné využít tak, že při kontrole, zda v *Port map* na určité pozici je nebo není platný ukazatel, nemusí být zámek. Bez tohoto zámku mohou nastat dvě situace:

1. Je vyčten neplatný ukazatel. To znamená, že pro port, který paket používá není registrován žádný funkce pro překlad aplikačních dat a pak se může předa dalšímu procesnímu bloku.

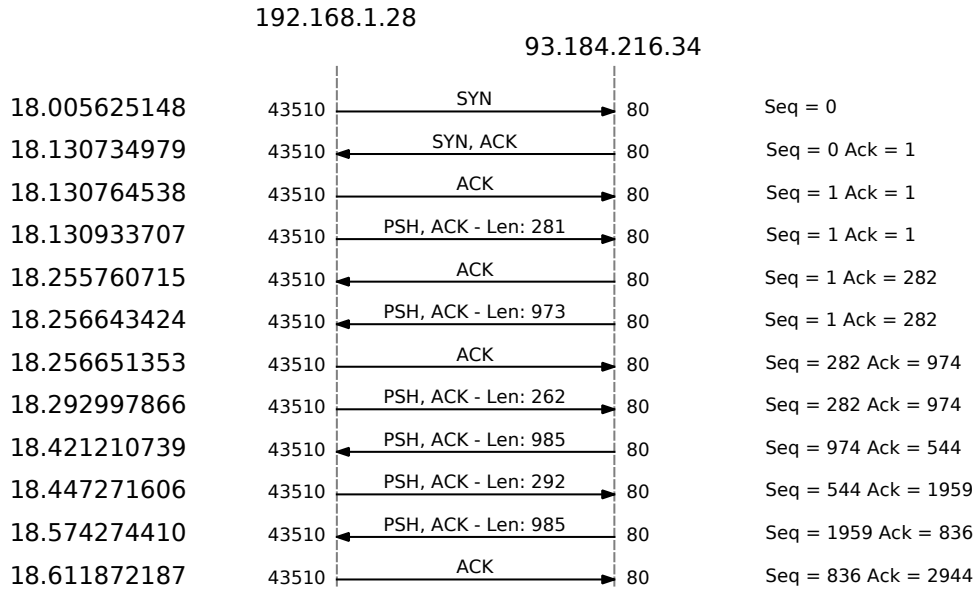
2. Je vyčten platný ukazatel. V tomto případě existuje zřetězený seznam, který je potřeba projít a aplikovat funkce pro překlad adres v aplikačních datech.

V druhém případu může nastat situace, kdy se začne procházet seznam, ale jiný proces ho mezitím odstraní. Pro tento případ je průchod seznamem chráněn zámkem, který umožňuje více čtecích vláken. Při čekání na vstup do kritické sekci, pak může dojít ke smazání seznamu, nad kterým má být proveden průchod. Toto je řešeno tak, že po získání zámku je znovu vyčten ukazatel z *Port map* a provedena kontrola na validitu ukazatele.

5.2 TCP mangling

TCP protokol pro zajištění spolehlivého protokolu zasílá v hlavičkách paketu sekvenční a potvrzovací číslo [28]. Při zahájení komunikace je sekvenční číslo inicializováno na pseudonáhodnou hodnotu. Každý paket, který je poslán v rámci tohoto spojení zvýší sekvenční číslo o velikost, která odpovídá počtu bajtů uložených v aplikační vrstvě, ale toto nové číslo

bude použito až v následujícím odeslaném paketu. Pokud paket obsahuje příznak SYN nebo FIN, pak se sekvenční číslo zvyšuje o jedna. Jakmile paket dorazí k cíli, příjemce odešle paket s nastaveným příznakem ACK a v poli potvrzovacího čísla bude uvedeno sekvenční číslo přijatého paketu zvýšeného o velikost aplikačních dat.



Obrázek 5.2: Příklad TCP spojení

Pomocí nástroje Wireshark byla zachycena komunikace s webovým serverem na doméně *example.com*, následně by z toho spojení byl vygenerován, který ilustruje použití sekvenčních čísel. Tento graf je na obrázku 5.2. Komunikace začíná paketem s příznakem SYN a sekvenčním číslem 0[2]. Příznak SYN zvyšuje sekvenční číslo o jedna a tato změna se projeví v potvrzovacím číslu následujícího paketu. V druhém paketu vzniká druhý paketový tok, který má vlastní sekvenční čísla. Třetí paket zakončuje 3-way handshake a obsahuje sekvenční číslo 1 (kvůli příznaku SYN v prvním paketu) a potvrzovací číslo 1 (kvůli příznaku SYN v druhém paketu). Ve čtvrtém paketu byly odeslány aplikační data s délkou 281 bajtů, ale sekvenční i potvrzovací číslo zůstává stejné. Změna sekvenčního čísla se projeví až v následujícím odchozím paketu (v příkladu se jedná o 7. paket). Patý paket potvrzuje přijetí předchozí zprávy, tím že nastavuje potvrzovací číslo na sekvenční číslo přijatého paketu (1) plus velikost jeho aplikačních dat (281), celkem tedy na 282.

Při překladu adres se může stát, že ALG algoritmus úpravou obsahu aplikačních dat změní celkovou velikost paketu. To se pak musí projevit i v sekvenčních a potvrzovacích číslech, jinak by mohlo dojít k rozpadnutí spojení. Opravu sekvenčních a potvrzovacích čísel řeší v NAT DPDK *TCP mangling*.

Pokud ALG algoritmus upraví paket tak, že se změní jeho velikost (a v paketu je použit TCP protokol), pak musí zajistit registraci toku, do kterého spadá daný paket, do *TCP manglingu*. Při registraci kromě toku se přidává rozdíl velikosti (v bajtech) mezi přijatým a upraveným paketem a očekávané sekvenční číslo v následujícím paketu daného toku. Pokud je v rámci stejného toku upraveno více paketů, pak s každou úpravou je o změně *TCP mangling* informován a celková velikost provedených úprav se akumuluje.

[2]Ve skutečnosti bylo použité sekvenční číslo rovno 3654209527, ale v příkladu jsou uvedená čísla relativní k této hodnotě.

TCP mangling používá hashovací tabulku pro sledování upravených toků. Klíčem do této tabulky je uspořádaná čtveřice (zdrojová IP adresa, cílová IP adresa, zdrojový TCP port, cílový TCP port), pro zdrojovou IP adresu a port jsou použity externí asociované adresy. Záznamy hashovací tabulky jsou ukazatele na kruhový buffer.

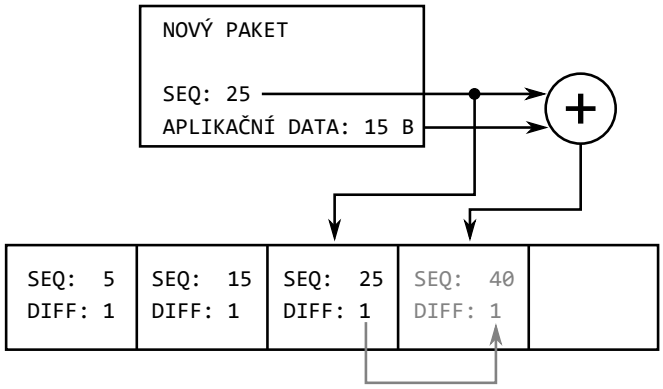
Při přidání nového toku do *TCP manglingu*, kruhový buffer obsahuje jediný záznam s očekávaným sekvenčním číslem následujícího paketu z daného toku a diferenci, což je velikost o kolik se má sekvenční číslo upravit. Při příchodu paketu z daného toku mohou nastat následující situace:

1. Doručený paket má sekvenční číslo shodné s tím, které je v naposledy přidaném záznamu kruhového bufferu.
2. Doručený paket má shodné číslo s číslem, které v kruhovém bufferu je, ale záznam není poslední přidaný.
3. Sekvenční číslo je větší, než číslo na poslední pozici v kruhovém bufferu.
4. Sekvenční číslo v doručeném paketu nemá shodu s žádným záznamem z kruhového bufferu.

V prvním a druhém případě se vždy upraví sekvenční číslo o diferenci, která je uložena v odpovídajícím záznamu. Ve čtvrté situaci se použije diference ze záznamu, který obsahuje největší sekvenční číslo, které je menší, než číslo v doručeném paketu.

Třetí situace nastane pouze tehdy, když odesílatel odešle více paketů za sebou, ale po cestě k uzlu s NAT DPDK jsou některé zahozeny (případně je některý síťový prvek prohodí). Pokud se detekuje takováto situace, pak musejí být všechny pakety, které mají vyšší sekvenční číslo, než je uloženo v kruhovém bufferu pozdrženy. K tomuto pozdržení dochází, protože paket, který na NAT DPDK nebyl doručen, by mohl být upraven ALG algoritmem. To by mohlo vést k úpravě velikost tohoto paketu a tedy ke změně diferencí pro po něm následujících paketů.

Nový záznam je v kruhovém bufferu vytvořen pouze v prvním případě. Nový záznam obsahuje očekávané sekvenční číslo následujícího paketu (tzn. součet sekvenčního čísla v přijatém paketu plus velikost aplikačních dat). Diference v novém záznamu je stejná jako v předposledním záznamu. Pokud se zjistí, že právě přijatý paket je upraven ALG algoritmem, pak je k diferenci v posledním záznamu bufferu přičtena diference nové úpravy.



Obrázek 5.3: Přidání záznamu do kruhového bufferu

Příklad přidání záznamu do kruhového bufferu je na obrázku 5.3. Příchozí paket má sekvenční číslo 25 a velikost aplikačních dat 15 B. Sekvenční číslo uložené v posledním záznamu kruhového bufferu je shodné s číslem v doručeném paketu. Jedná se tedy o očekávaný paket. Do kruhového bufferu je tedy přidán záznam se sekvenčním číslem 40 $(25 + 15)$ a diference je přenesena z předchozího záznamu. Paket po provedeném *TCP manglingu* pak bude mít sekvenční číslo 26 (původní sekv. číslo + diference).

V této chvíli jsou vyřešeny sekvenční čísla paketů, není se budeme zaobírat změnou potvrzovacího čísla. Pakety obsahující potvrzovací čísla k určitému toku mají zaměněné zdrojové a cílové adresy, proto pro opravu potvrzovacích čísel se do hashovací tabulky *TCP manglingu* používá klíč (cílová IP adresa, zdrojová IP adresa, cílový TCP port, zdrojový TCP port).

V kruhovém bufferu jsou uloženy pouze sekvenční čísla a diference, proto se musí při vyhledávání korespondujícího záznamu k danému potvrzovacímu číslu nejdříve sečíst uložená diference s uloženým sekvenčním číslem. Po sečtení je možné provést porovnání, zda daný záznam vyhovuje danému potvrzovacímu číslu. Pokud se dále v textu objeví potvrzovací číslo v kontextu se záznamem v kruhovém bufferu, pak je pro daný záznam myšlen součet reálně uloženého sekvenčního čísla a uložené diference.

Při příchodu paketu s potvrzovacím číslem mohou při vyhledávání odpovídajícího záznamu v kruhovém bufferu nastat tyto čtyři situace:

1. V kruhovém bufferu existuje záznam, který přesně odpovídá potvrzovacímu číslu.
2. Potvrzovací číslo je větší, než největší uložené potvrzovací číslo.
3. Potvrzovací číslo je větší, než nejmenší uložené potvrzovací číslo, ale neexistuje přesná shoda.
4. Potvrzovací číslo je menší, než nejmenší uložené potvrzovací číslo.

Nejdříve vylučme druhý případ, protože ten teoreticky nastat nemůže (nemůže přijít potvrzení přijetí dat, která ani nebyla odeslána). Pokud takovýto paket dorazí, pak se pravděpodobně jedná o útok, proto je paket přeposlán bez dalších úprav.

Pokud nastane první situace, pak je od potvrzovacího čísla v doručeném paketu odečtena diference (pokud v odeslaném paketu bylo k sekvenčnímu číslu přičtena hodnota x, pak v příchozím paketu musí být od potvrzovacího čísla hodnota x odečtena). Následně se odstraní z kruhového bufferu všechny položky před nalezeným záznamem (implementačně se jedná pouze o posunutí ukazatelů). Toto odstranění je možné, protože zpracovávaný paket potvrzuje, že veškeré pakety, kterým odpovídají sekvenční čísla ze smazaných záznamům, až po ten nalezený, a to včetně něj, byla úspěšně doručena. Důvod, proč jsou odstraněny pouze záznamy před nalezeným záznamem, je z možné přehození a nebo znovu-zaslání potvrzujících paketů. Pokud by došlo k prohození pořadí potvrzovacích paketů, před tím, než dorazí na NAT DPDK a tedy nastane čtvrtá situace, pak stačí v těchto paketech přepsat potvrzovací číslo, podle prvního uloženého záznamu.

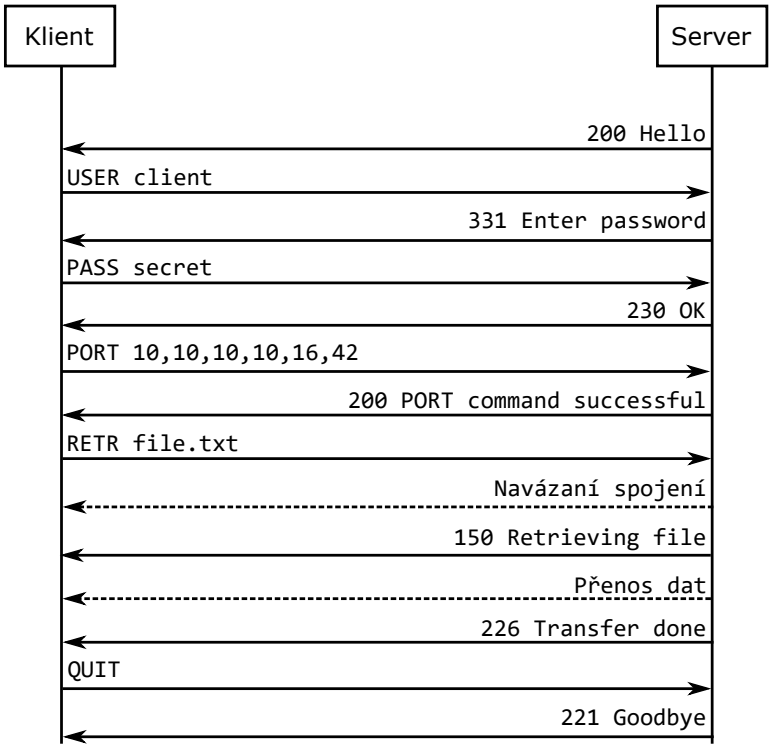
Pokud nastane třetí situace, pak je vyhledán záznam, který má největší potvrzovací číslo, ale menší, než je uložena v doručeném paketu. Následně se postupuje jako v prvním případě. Nevýhoda tohoto přístupu je, že se sníží počet potvrzených bajtů a odesílatel, tak při znovu-zaslaní je pošle zbytečně. Prakticky tato situace nenastane, protože by to znamenalo, že příjemci došel paket pouze z části, což se v TCP/IP sítích nestává. Teoreticky je tato varianta možná a proto je pro ní implementována podpora.

5.3 FTP

File Transfer Protocol(FTP) je určen pro přenos souborů mezi počítači. Protokol je specifikován ve standardu RFC 959 [19]. FTP je založeno na modelu klient-server a pro komunikaci používá dva porty. První port se používá pro navázání spojení se serverem, které se nazývá kontrolní. Druhý port je použit pro vytvoření tzv. datového spojení, kterým se přenáší samotné soubory. Protokol použitý v kontrolním spojení používá čitelný text.

Klient odesílá požadavky na server přes kontrolní spojení (standardně FTP server naslouchá na TCP portu 21). Tyto požadavky musí odpovídat specifikaci v RFC 959 [19]. Server na ně odpovídá formou třímístných kódů, které mohou být následovány textovou zprávou.

Datové spojení může být vytvořeno v aktivním nebo pasivním režimu. V aktivním režimu je serveru odeslána zpráva, která obsahuje IP adresu a port na kterou se má server připojit. Server se na danou IP adresu a port pokusí připojit a o výsledku informuje klienta. Následně je možné přes datový kanál přenášet soubory, případně jiné informace, které není možné přenést přes kontrolní spojení (například seznam souborů na straně serveru).

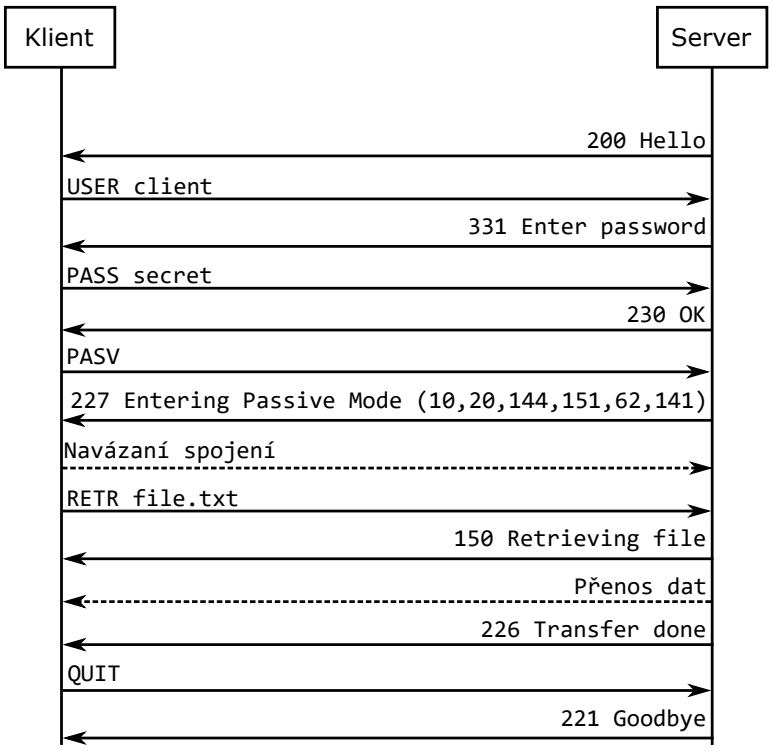


Obrázek 5.4: FTP – aktivní režim

Příklad aktivního spojení je naznačen na obrázku 5.4. Prvních pět zpráv slouží pro autentizaci. Šestá zpráva, příkaz `PORT`, oznamuje serveru, že se má připojit na IP adresu 10.10.10.10 a port 4138 (číslo 42 je hodnota v desítkové soustavě, která udává 8 spodních bitů portu a číslo 16 horních 8 bitů). Následuje potvrzení příkazu `PORT` a vyžádání souboru `file.txt` ze serveru. Přerušovanou čárou je zobrazeno datové spojení, které je vytvořeno

po příkazu `RETR` (zdrojový TCP port je standardně 20). V dalším kroku server oznamuje klientovi úspěšné navázání datového spojení a odesílá data. Skončení přenosu je oznámeno klientovi a klient posílá příkaz k uzavření kontrolního spojení.

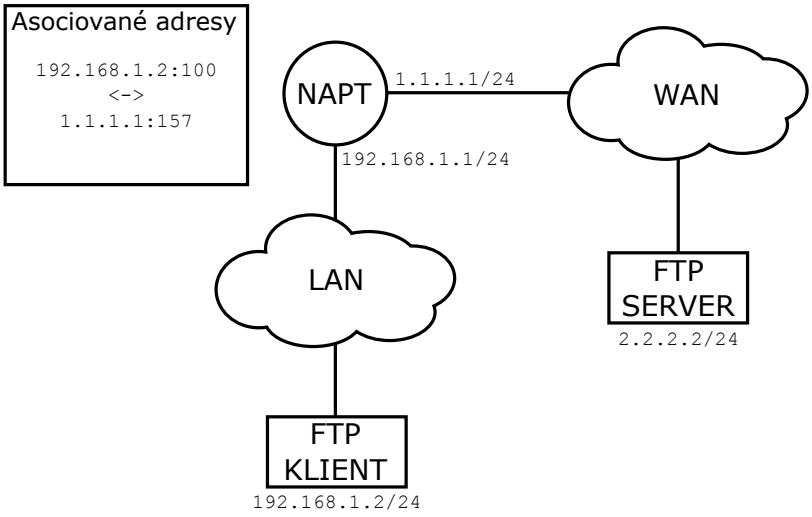
Pasivní spojení funguje naopak, než aktivní. Klient se serveru zeptá na jakou IP adresu a port se má připojit. Server odešle v odpovědi požadované informace, klient se připojí a tím je vytvořeno datové spojení.



Obrázek 5.5: FTP – pasivní režim

Příklad na obrázku 5.5 je stejný jako v případě obrázku 5.4, ale popise přenos souboru v pasivním režimu. Prvních pět zpráv je stejných jako v minulém příkladu. Následuje žádost o vytvoření datového spojení v pasivním režimu (příkaz `PASV`). Server odpovídá s kódem 227 a součástí zprávy je IP adresa a port (10.20.144.151:16013) na kterou se má klient připojit. Klient naváže spojení a požádá server o přenos souboru `file.txt`. Server žádost potvrdí, přenese soubor, oznámí klientovi ukončení přenosu a klient uzavře kontrolní spojení.

Nejdůležitější rozdíl pro překlad adres je strana, která navazuje datové spojení, u aktivního jej navazuje server a u pasivního klient. Předpokládejme, že FTP klient je uvnitř lokální domény. V případě pasivního režimu je FTP protokol v NAT DPDK implicitně podporován, protože všechny požadavky jsou navazovány z lokální domény do domény externí. V případě aktivního spojení to už neplatí.



Obrázek 5.6: FTP – topologie

Předpokládejme topologii na obrázku 5.6. Uvažujme situaci, kdy FTP klient vytvořil kontrolní spojení s FTP serverem a má vytvořenou asociaci `192.168.1.2:100` na `1.1.1.1:157`. Nyní klient chce přijmout soubor a pošle serveru příkaz `PORT 192,168,1,2,16,42`. Server se pokusí navázat spojení s `192.168.1.2:4138`, toto spojení se mu však nepodaří vytvořit, protože nezná cestu do sítě `192.168.1.0`.

Řešení tohoto problému je následující. Pokud na NAPT přijde paket, který směřuje na TCP port 21 a obsah paketu je validní FTP příkaz, pak stačí sledovat komunikaci a v případě příkazu `PORT` provést následující operace:

1. Získat z FTP příkazu IP adresu a port.
2. Provést kontrolu, zda se zdrojová IP adresa paketu shoduje s adresou uvnitř příkazu `PORT`. Jestliže se neshodují, pak se může jednat o útok na NAPT, ale stejně tak to může být validní požadavek. Z důvodu, že se nejedná o častý jev, je to považováno za útok a paket je odeslán bez úprav.
3. Pokud se shodují, pak je do asociační tabulky přidán nový záznam a obsah příkazu `PORT` je dle toho adekvátně upraven.
4. Provést TCP mangling
5. Opravit velikost paketu uvedenou v IP hlavičce.
6. Opravit kontrolní součet paketu.
7. Předa paket dál v NAT DPDK.

Pro TCP i UDP platí, že po změně paketu je nutné znovu vypočítat kontrolní součet v TCP/UDP hlavičce, ale i v IP hlavičce. V případě, že se jedná o malou změnu a celková velikost paketu není změněna je možné provést doplňkový výpočet kontrolního součtu, který není výpočetně náročný jako jeho celý výpočet. Protože FTP protokol je však textový, tak je

možné, že s přepsáním IP adresy a portu dojde ke změně velikosti. Například příkaz `PORT 10,10,10,10,10,10` může být změněn na `PORT 1,1,1,1,1,1`, tento paket je pak menší o 6 B. V tomto případě se už dopočítává celý kontrolní součet.

5.4 SIP

SIP (z angl. Session Initiation Protocol) je aplikační protokol, který umožňuje vytvářet, upravovat a ukončovat multimediální relace, jako například telefonní hovory. Standardně se používá UDP port 5060, ale může být použit i TCP port 5060. SIP má dvě verze, první je popsána v RFC 2543 [9] a druhá v RFC 3261 [23]. V této práci se budeme zabývat převážně druhou verzí.

SIP je, stejně jako FTP, textový protokol. Syntaxe a sémantika protokolu SIP je postavena na tzv. *Internet Message Format*, který je definován v RFC 2822 [21]. Na stejném formátu zpráv je postaven například také protokol HTTP.

Adresování v protokolu SIP je definováno pomocí tzv. SIP URI, které se skládá ze tří částí. První je řetězec `sip:`, za ním následuje jméno uživatele a poté je doména, kde se uživatel nachází. Jméno uživatele a doména se odděluje znakem `@`. Doména může být jméno, které je možné přeložit pomocí DNS, nebo přímo IP adresa. Příkladem SIP URI je `sip:1-999-123-4567@voip-provider.example.net`.

Jedna SIP zpráva se skládá maximálně ze tří částí. Jedná se o typ zprávy, za ní jsou hlavičky zprávy a po hlavičkách může následovat tělo zprávy. Typ a hlavičky jsou součástí každé SIP zprávy. Tělo obsahují pouze některé zprávy. SIP zprávy se rozdělují na dva základní typy:

- požadavek (angl. request)
- odpověď (angl. response).

Požadavky mohou obsahovat jednu z následujících metod:

REGISTER
: Tato metoda slouží k registraci uživatele k SIP registrátoru. SIP registrátor je server, který registrační zprávu zpracuje a uloží do tzv. *Location service*. *Location service* je databáze, která umožňuje provádět směrování SIP zpráv mezi více doménami.

INVITE
: Zpráva `INVITE` slouží k oznámení, že odesílat si přeje komunikovat s adresátem, který je uveden v hlavičkách zprávy.

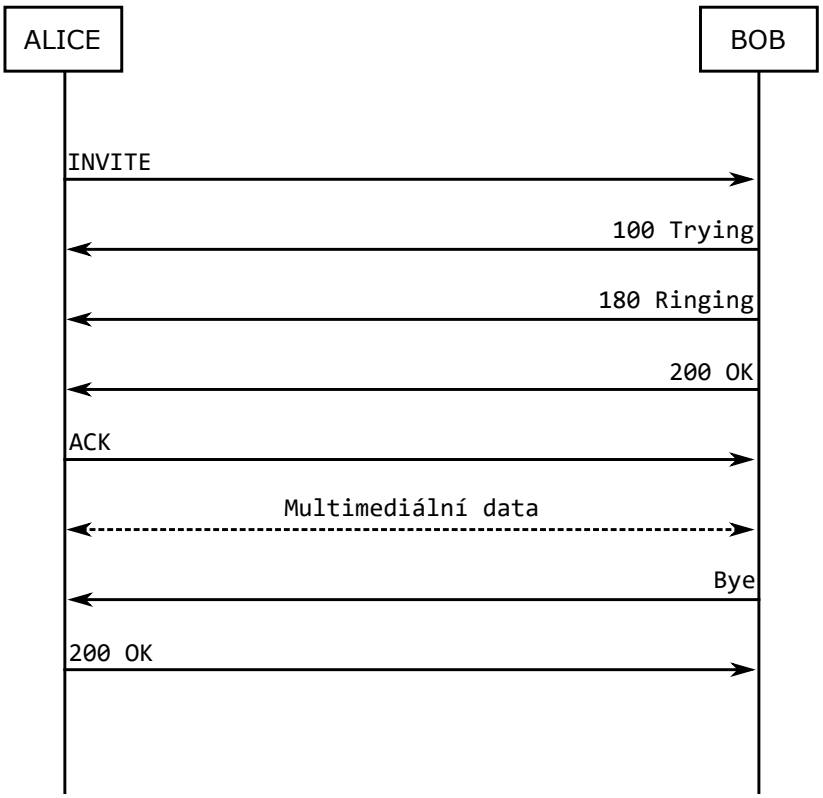
ACK
: Tato metoda slouží pro potvrzení navázání spojení.

CANCEL
: `CANCEL` je odeslán, pokud jedna z komunikujících stran chce zrušit některý předešlý požadavek.

BYE
: Zpráva, která se odesílá v případě žádosti o ukončení spojení.

OPTIONS
: Tato zpráva umožňuje zjistit jaké funkce podporuje příjemce.

Odpovědi na zprávy typu žádost obsahují třímístný číselný kód, který odpovídá stavu vykonávání požadavku. V některých případech může být odesláno více odpovědí na jeden požadavek.



Obrázek 5.7: SIP – příklad spojení mezi dvěma klienty

Na obrázku 5.7 je příklad VoIP spojení mezi dvěma klienty Alicí a Bobem. Alice se chce spojit s Bobem a tak její VoIP klient odešle telefonu Boba požadavek s metodou `INVITE`. Telefon Boba pošle odpověď Alici, kterou informuje, že se snaží vytvořit hovor. Následně Bobův telefon pošle další odpověď Alici, kde jí oznamuje, že telefon vyzvání. Ve chvíli, kdy Bob telefonát přijme, jeho telefon o tom informuje klienta Alice (zpráva `200 OK`). Klient Alice nakonec pošle požadavek Bobovi, kterou potvrzuje jeho poslední odpověď (`200 OK`). Součástí předchozích zpráv byly informace, které určili jaký má být použit protokol pro multimediální stream, jaký má být jeho formát, jaké IP adresy a porty se mají použít pro spojení. Pomocí těchto informaci je spojení vytvořeno. Nakonec, když Bob zavěsí telefon tak jeho klient informuje Alici zasláním požadavku `BYE`. Aličin klient pouze potvrdí přijetí požadavku a hovor je ukončen.

Podívejme se jak konkrétně vypadal požadavek Alice:

```
INVITE sip:bob@1.1.1.1 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.10;branch=hifeU8498
To: Bob <sip:bob@1.1.1.1>
From: Alice <sip:alice@provider.net>;tag=32549854
Call-ID: ncjehvaw@10.10.10.10
CSeq: 12 INVITE
Contact: <sip:alice@10.10.10.10>
Content-Type: application/sdp
Content-Length: 326
```

První řádek označuje typ zprávy, v tom případě se jedná o požadavek s metodou `INVITE`. Druhý řádek (hlavička `Via`) určuje cestu vsíti pro odpověď na tuto zprávu. Hlavička `To` určuje adresáta zprávy, hlavička `From` identifikuje odesílatele. `Call-ID` je identifikátor dialogu, jinak řečeno identifikátor relačního toku. `CSeq` je sekvenční číslo požadavku. `Contact` obsahuje přímou cestu k iniciátorovi spojení. `Content-Type` specifikuje v jakém formátu je obsah těla (v příkladu není tělo uvedeno) a `Content-Length` informuje o tom jak velké je tělo zprávy (hodnota je uvedena v bajtech).

Pro navázání multimediálního spojení je potřeba definovat, jak spojení vytvořit a určit jeho obsah. K tomuto účelu souží tělo SIP zprávy. Jedním z protokolů, které se v těle zpráv používají je SDP (z angl. Session Description Protocol) protokol, ten je definován v RFC 4566 [10]. Základní strukturu si uvedeme na příkladu. Následující text je tělo SIP zprávy, kterou poslala Alice Bobovi ve zprávě `INVITE`.

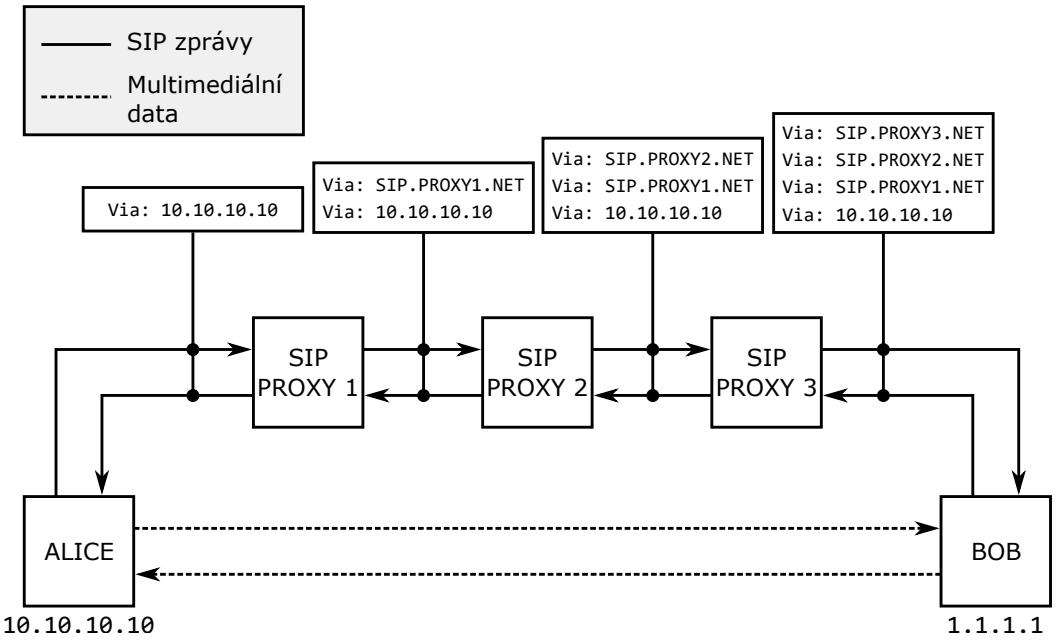
```
v=0
o=debian3 2457 504 IN IP4 10.10.10.10
s=Talk
c=IN IP4 10.10.10.10
t=0 0
m=audio 7078 RTP/AVP 124 111 110 0 8 101
a=rtpmap:124 opus/48000
a=fmtp:124 useinbandfec=1; usedtx=1
a=rtpmap:111 speex/16000
a=fmtp:111 vbr=on
a=rtpmap:110 speex/8000
a=fmtp:110 vbr=on
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-11
```

Každý řádek v SDP protokolu začínám jedním písmenem (označme ho deskriptor), které definuje syntaxi a sémantiku výrazu za znaménkem rovná se. V uvedeném příkladu nejsou zobrazeny všechny možné deskriptory, ale pro překlad síťových adres je výčet dostačující. Deskriptor `v` definuje verzi SDP protokolu (aktuálně existuje pouze verze 0). Deskriptor `o` je zkratka z angl. *originator* a popisuje iniciátora spojení. Součástí deskriptoru `o` je IP adresa iniciátora, ale tento řádek slouží pouze pro jeho identifikaci, a nikoliv pro vytváření multimediálního spojení, z tohoto důvodu se tato adresa nepřekládá. Deskriptor `c` oznamuje, na které IP adrese iniciátor očekává příchozí multimediální data, v tomto případě se adresa při překladu přeložit musí, jinak by příjemce zprávy odesílal data do (z jeho pohledu)

nedostupné sítě. Poslední významný deskriptor pro překlad adres je `m`. Tento deskriptor popisuje použitý protokol pro přenos multimediální dat (v příkladu je to RTP), o jaký typ dat se jedná (audio) a na kterém TCP/UDP portu se spojení očekává (v případě RTP se jedná o UDP a port je v příkladu 7078) ostatní hodnoty a zbylé nepopsané deskriptory definují další atributy spojení, které pro překlad adres nejsou podstatné.

V příkladu na obrázku 5.7 se Alice snaží spojit přímo s Bobem. SIP umožňuje vytvořit spojení, u kterého se SIP zprávy nezasílají přímo příjemci, ale pomocí tzv. SIP proxy. Například pokud v SIP URI příjemce není IP adresa, ale doménové jméno, tak DNS záznam může obsahovat IP adresu SIP proxy, který zajistí přeposlání zprávy blíže k příjemci.

Komunikace, kde se iniciátor snaží spojit přes více SIP proxy, je na obrázku **??**. Pokud jde zpráva od Alice (iniciátora) k Bobovi (příjemci) přes SIP proxy, pak každý SIP proxy ke zprávě přidá vlastní hlavičku `Via`. Jakmile Bob zpracuje požadavek, tak do hlavičky odpovědi přidá veškeré hlavičky `Via`, které byly v příchozí zprávě. odpověď následně zašle na adresu, která je uvedena v pořadí první hlavičce `Via`. SIP proxy po zpracování odpovědi odstraní svoji hlavičku a odpověď přepošle na další adresu uvedenou v hlavičkách `Via`.



Obrázek 5.8: SIP – příklad se SIP proxy

Pokud je použito navázání spojení přes SIP proxy, pak inicátor zná pouze příjemcovu SIP URI, což může znamenat, že nezná jeho IP adresu, ani port, na který by se měl připojit. Tyto údaje získá, až ve chvíli, kdy na jeho zprávu `INVITE` přijde odpověď `200 OK`,jejíž součástí je tělo zprávy s potřebnými informacemi.

Pokud mají být překladem adres podporovány odchozí hovory, které používají signalizační protokol SIP, pak je potřeba upravit některé zprávy. V SIP požadavku s metodou `INVITE` je potřeba provést následující operace:

1. Pokud hlavička `Via` obsahuje IP adresu z lokální domény, pa kse nahradí za adresu, která je k použité adrese asociována.
2. Zpracovat SIP hlavičku `Contact` a stejně jako u hlavičky `Via` nahradit lokální adresu za externí.
3. Alokovat novou IP adresu a port pro spojení, které bude použito pro multimediální data[3].
4. Nahradit v SDP deskriptoru `c` IP adresu z lokální domény za nově alokovanou adresu.
5. Nahradit port v SDP deksriptoru `m` za nově alokovaný.
6. Opravit hodnotu v SIP hlavičce `Content-Length` tak, aby odpovídala velikosti upraveného těla zprávy.
7. Upravit délku paketu uloženou v IP hlavičce.
8. Pokud je použit UDP protokol upravit délku paketu uloženou v UDP hlavičce.
9. Pokud je použit TCP protokol provést TCP mangling.
10. Opravit kontrolní součty v hlavičkách.

Jakmile dorazí odpověď na požadavek provedou se následující operace:

1. Externí adresa v hlavičce `Via` se nahradí za asociovanou lokální adresu.
2. Opraví se SIP hlavička `Contact` do původního stavu.
3. Pokud se jedná od odpověď `200 OK`:
   (a) Zpracuje se tělo zprávy a získá se IP adresa a port, na kterou se má iniciátor připojit.
   (b) Vytvoří se záznam v tabulce *Elements*, který obsahuje asociaci adres alokovaných při zpracování požadavku (předchozí seznam, krok 3).
   (c) Přidá se ukazatel do *LAN* $\rightarrow$ *WAN NAT table* a *WAN* $\rightarrow$ *LAN NAT table* na nový záznam v *Elements*.
4. Upraví se délka paketu uložená v IP hlavičce.
5. Pokud je použit UDP protokol upraví se délka paketu uložená v UDP hlavičce.
6. Pokud je použit TCP protokol provede se TCP mangling.
7. Opraví se kontrolní součty v hlavičkách.

[3] Alokace je provedena pouze pokud se jedná o nový požadavek. Pokud se jedná o znovu zaslaný požadavek, pak se použijí původně alokované adresy

Záznam do asociačních tabulek (krok 5c) se přidává až ve chvíli, kdy je známa reálná adresa příjemce, protože jeho IP adresa a použitý port je součástí klíče do těchto tabulek.

Výše popsaná technika funguje pouze v případě, kdy je použit jeden *worker*, nebo SIP zprávy jsou posílány přímo příjemci bez SIP proxy. Z kapitoly 4.2.2, *distributor* provádí rozdělování příchozích paketů na *workery* podle IP adres uzlů v externí doméně a každý *worker* má vlastní asociační tabulky. Nyní si představme situaci, jako je znázorněna na obrázku 5.8. Pokud NAT DPDK používá více než jeden *worker*, pak odchozí SIP zprávy zpracovává jeden *worker*, ale spojení, které je vytvořeno pro multimediální data zpracovává druhý *worker*. Takže při zpracovávání odpovědi, když je vytvořen záznam do asociačních tabulek (krok 5c), tak je vytvořen do tabulek prvního *workeru*, ale měl by být ve druhém. V aktuální implementaci NAT DPDK není mezi *workery* žádná synchronizace a tedy z jednoho *workeru* není možné zasáhnout do asociační tabulky druhého *Workeru*.

Součástí NAT DPDK je rozhraní, které umožňuje přidávat některá pravidla za běhu systému. Toto rozhraní je možné využít k přidání dočasného DNAT pravidla. V novém pravidlu se nastaví, že je očekáváno spojení na konkrétní port iniciátora (ten je známý z požadavku `INVITE`), ale zdroj je zatím neznámý. Následně se zkoumá každý paket, který přijde na NAT DPDK a vyhoví přidanému pravidlu. Pokud zatím není známá adresa příjemce, pak jsou všechny tyto pakety zahazovány. Ve chvíli, kdy dojde odpověď `200 OK` a příjemce je znám počká se na jeho první paket, nebo na první paket iniciátora, který používá stejný port pro komunikaci v opačném směru. V této chvíli je vytvořen standardní záznam v asociační tabulce *workeru*, na který paket dorazil a smazáno dočasné DNAT pravidlo.

Aktuální implementace pokrývá pouze odchozí hovory. Pro příchozí hovory je potřeba do NAT DPDK naimplementovat celého SIP registrátora a SIP proxy. Tato implementace je nad rozsah této práce, ale při vytváření algoritmu pro zpracovávání SIP zpráv byl na toto brán ohled a zpracování celého SIP protokolu je připraveno na další rozšíření.

5.5 IRC DCC

IRC DCC (z angl. Internet Relay Chat Direct Client-to-Client) rozšiřuje protokol IRC, který je definován v RFC 1495 [17]. Tento protokol umožňuje vytvořit spojení mezi dvěma klienty za využití IRC serveru. IRC DCC není nikde standardizované a vytvořili ho lidé, kteří implementovali IRC klienta ircII [2].

Pro zaslání příkazu konkrétnímu klientu se v používá příkaz `PRIVMSG`, který je součástí standardu IRC. `PRIVMSG` má následující syntaxi:

```
PRIVMSG <cíl>[,<cíl>,...] :<text zprávy>
```

Cíl je uživatelské jméno nebo název kanálu kam je příkaz určen. Pro rozlišení mezi textem zprávy určeného pro klienta a příkazem DCC, jsou DCC příkazy obklopeny symbolem, který má hodnotu `0x01`. V textu se pro tento symbol běžně používá označení `^A`.

IRC DCC se používá ve dvou situacích. První je vytvoření soukromého chatu, kdy uživatelé nechtějí, aby jejich zprávy, byly nejdříve odeslány na IRC servery a až pak k jejich příjemci. Druhé využití je pro přenos souborů.

Příkaz pro zahájení DCC chatu je:

```
PRIVMSG <uživatel> :^ADCC CHAT chat <ip> <port>^A
```

a podobně pro zaslání souboru:

```
PRIVMSG <uživatel> :^ADCC SEND <soubor> <ip> <port> [velikost souboru]^A
```

Na rozdíl od prokolu SIP jsou IP adresy vloženy ve formě jednoho desítkového čísla (všechny 4 bajty IP adresy jsou interpretovány jako jedno 32-bitové číslo). V obou případech iniciátor (ten co zaslal `PRIVMSG`) čeká, až se cílový uživatel připojí na vloženou IP adresu a port.

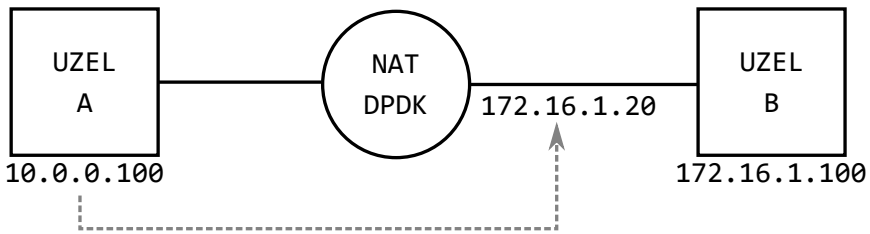
Pro překlad adres je potřeba provést následující kroky:

1. Zpracovat DCC příkaz a získat zaslanou IP adresu a port.
2. Alokovat IP adresu a port z externí domény.
3. Provést nahrazení IP adresy a portu v příkazu za nově alokované.
4. Vložit dočasné DNAT pravidlo (viz 5.4).
5. Provést TCP mangling
6. Opravit velikost paketu v IP hlavičce.
7. Opravit kontrolní součet paketu.

Kapitola 6

Testování a měření

Pro otestování zda je implementace ALG algoritmů korektní, byl použit framework NTF [6] (z angl. NAT Testing Framework). Tento nástroj umožňuje pomocí konfiguračních skriptů provést inicializaci celé síťové topologie. Následně na základě skriptů, které definují prováděné testy, spustí samotnou testovací proceduru. Nakonec vrátí výsledky, které popisují, jak provedené testy dopadly.



Obrázek 6.1: Testovací topologie

Pro testovací scénáře byla použita topologie na obrázku 6.1. Uzel A má přiřazenou IP adres `10.0.0.100`, která náleží do lokální domény. Tato adresa bude překládána na adresu `172.16.1.20`. V jednotlivých testech uzel A navazuje spojení s uzlem B. Způsob a protokol vytvářeného spojení závisí na jednotlivých testech.

6.1 Testování ALG pro FTP

Při testování ALG pro FTP uzel A vystupuje jako FTP klient, zatímco uzel B jako server. V provedeném testu se klient na uzlu A připojí na FTP server uzlu B. Následně si vyžádá přenos souboru z FTP serveru pomocí aktivního spojení. Po dokončení přenosu se klient od serveru odhlásí a porovná se MD5 hash staženého souboru s MD5 hashem souboru na FTP serveru. Pokud hashe souhlasí, pak je test prohlášen za úspěšný.

FTP protokol pro zajištění spolehlivého přenosu dat používá TCP protokol. Při změně příkazu `PORT` dojde ke zvětšení velikosti paketu. Tato vlastnost zároveň testuje funkčnost *TCP manglingu*.

Byly spuštěny dvě sady testů. V první sadě vždy klient navázal jedno spojení, získal

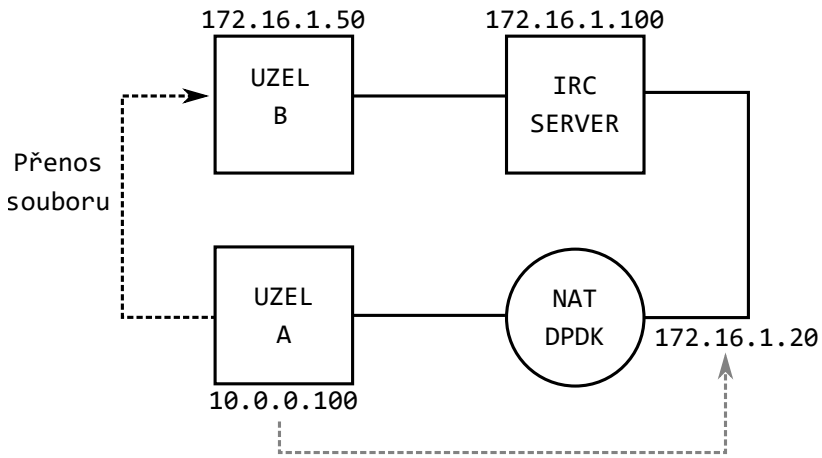
soubor ze serveru a porovnal hashe. To se opakovalo 30× za sebou. V druhé sadě se navázalo 1000 souběžných spojení, které provedly stažení souboru. Nakonec se porovnaly hashe souborů.

6.2 Testování ALG pro SIP

Při testování ALG pro SIP je na uzlu A i B nainstalován SIP klient *linphonec*. V testech se uzel A snaží navázat telefonní spojení s uzlem B. Test je označen za úspěšný, pokud jsou detekováno, že se podařilo navázat spojení pomocí protokolu RTP, který zajišťuje výměnu multimediálních dat.

6.3 Testování ALG pro IRC

Aktuálně není součástí NTF podpora pro testování IRC DCC protokolu. Z tohoto důvodu bylo testování provedeno manuálně. Použitá topologie ověřující funkčnost ALG pro IRC DCC je na obrázku 6.2.



Obrázek 6.2: Testovací topologie pro protokol IRC DCC

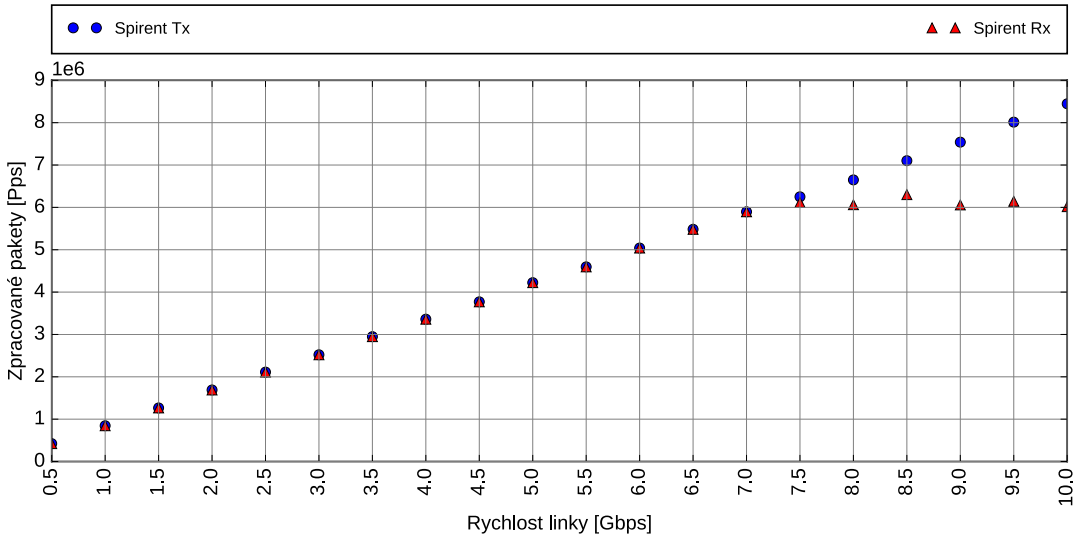
Na začátku testu se připojily uzly A a B k IRC serveru. Uzel A oznámil uzlu B, že mu chce zaslat soubor. Následně IRC klient uzlu A čekal na příchozí spojení od uzlu B. V okamžiku, kdy se uzel B připojil (pomocí informací uvnitř `DCC` příkazu) k uzlu A, tak uzel A začal odesílat soubor. Pro ověření, že test proběhl úspěšně, byl použit stejný způsob jako v případě FTP. Pokud MD5 hash přeneseného souboru odpovídal souboru na straně odesílatele, pak byl test prohlášen za úspěšný.

6.4 Měření

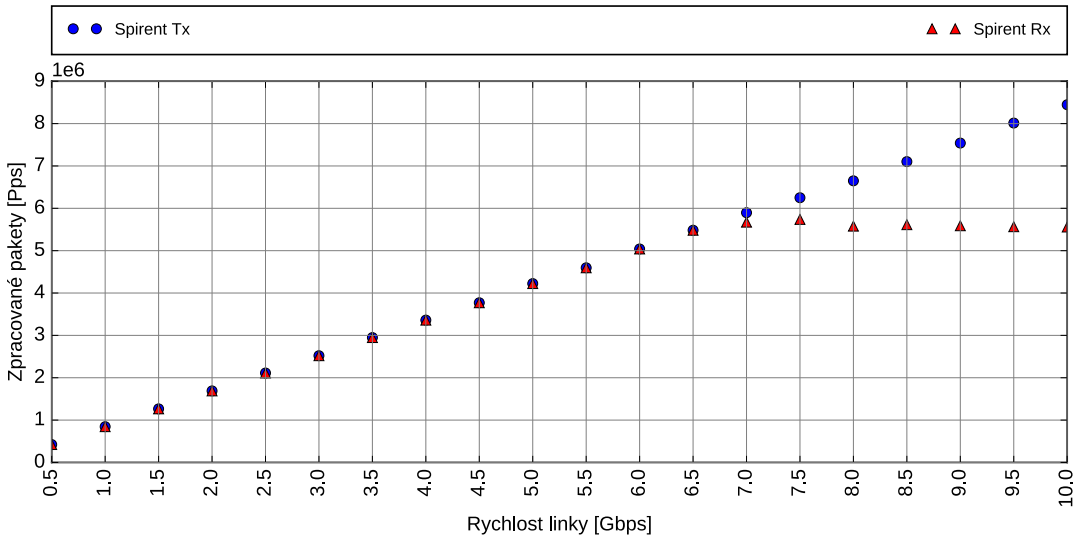
Součástí této práce je také změřit propustnost NAT DPDK. Měření probíhalo na počítači s procesorem Intel Xeon E5-2620. Všechny zobrazené grafy z měření NAT DPDK byly provedeny se 128 B pakety. Pro 512 B pakety všechna měření dosahovala plné propustnosti.

Měření byla provedena stejným způsobem jako v kapitole 3. Jediný rozdíl byl ve zvětšení kroku při zvyšování zátěže linky. Zatímco u předchozích testů se zátěž linky zvyšovala postupně o 1 % maximální saturace linky, tak zde byl krok zvýšen na 5 %. Tato změna byla provedena z důvodů snížení časové náročnosti na provedení jednotlivých měření, ale vypovídající hodnota měření zůstala zachována.

První měření bylo v režimu, kdy na jednom jádře procesoru běžel jak *distributor*, tak jeden *worker*. Výsledky pro variantu bez implementovaného rozhraní pro ALG jsou v grafu 6.3 a pro variantu s implementovaným ALG jsou v grafu 6.4. Zde je možné pozorovat, že přidání implementace ALG způsobilo snížení propustnosti NATu přibližně o 600 000 pps. Celková propustnost NATu (včetně implementace ALG) se pohybuje kolem 5,5 Mpps.

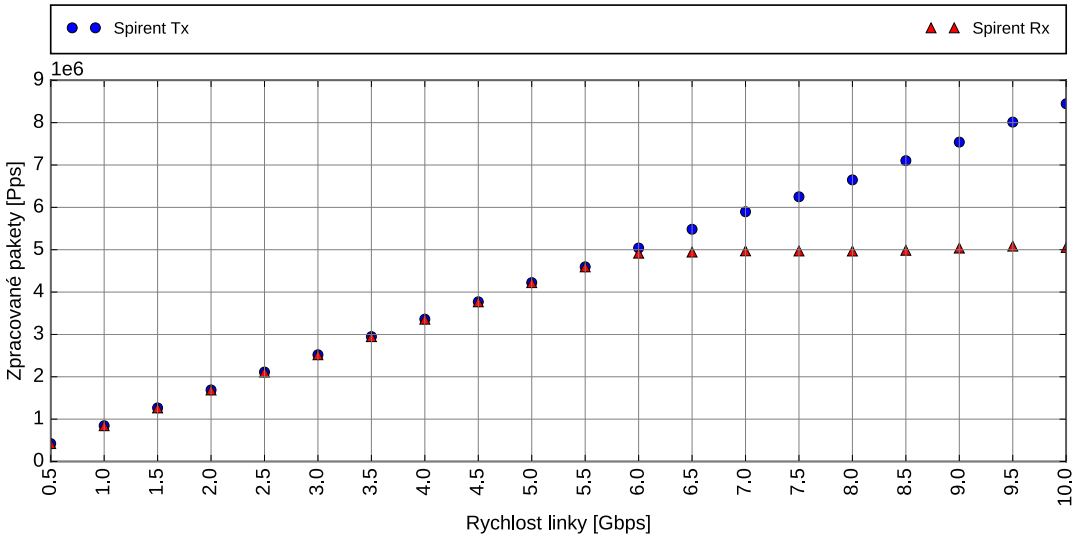


Obrázek 6.3: Intel Xeon – graf závislosti propustnosti na zátěži linky bez ALG, 128 B, 1 jádro



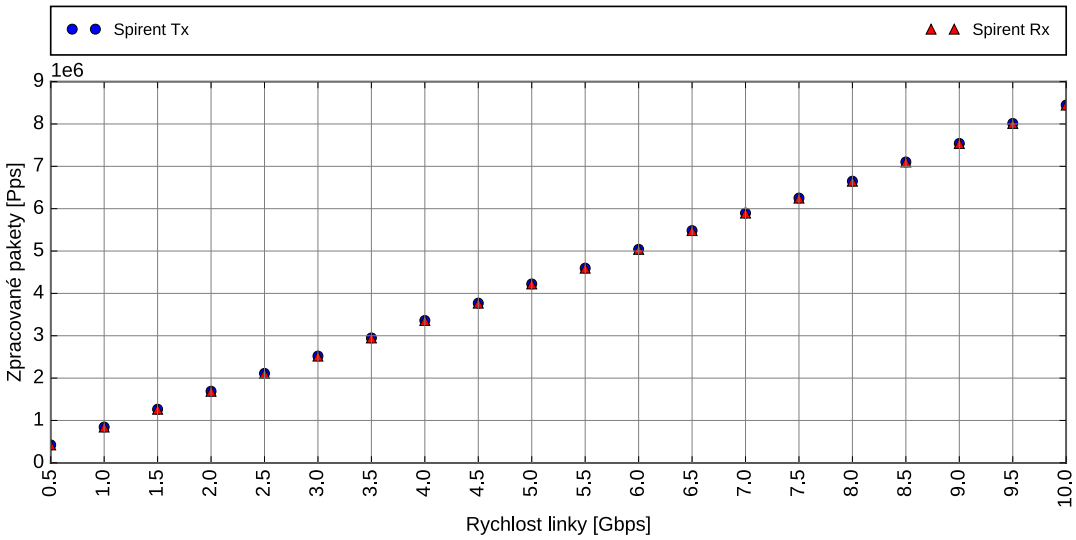
Obrázek 6.4: Intel Xeon – graf závislosti propustnosti na zátěži linky s ALG, 128 B, 1 jádro

Pokud se použije jedno jádro pro *distributor* a jedno pro *worker*, pak kvůli přidané komunikační režii maximální propustnost poklesne na přibližně 5 miliónů paketů za sekundu (viz obrázek 6.5).



Obrázek 6.5: Intel Xeon – závislosti propustnosti na zátěži linky, 128 B, 2 jádra

Při přidání dalšího *workeru* jsou všechny příchozí pakety zpracovány a odeslány. Toto chování je možné sledovat i při dalším přidávání jader. Výsledek tohoto měření s celkem třemi použitými jádry je na obrázku 6.6. Tohoto výsledku dosahuje implementace NATu, bez ALG rozšíření, ale i s ním. Pokud to shrneme, přidané ALG rozhraní snížilo celkovou propustnost jednoho jádra, ale při paralelním zpracování je přidaná režie dostatečně malá, aby se neprojevila.



Obrázek 6.6: Intel Xeon – závislosti propustnosti na zátěži linky, 128 B, 3 jádra

Kapitola 7

Závěr

S implementací překladu adres v OS Linux je možné dosáhnout až 2,5 miliónu zpracovaných paketů za sekundu. To může být dostačují v sítích, kde se průměrná velikost paketu pohybuje kolem 512 B. V sítích, kde je průměrná velikost paketu nižší než 512 B je vhodné nasadit specializované řešení. Jedno z těchto řešení je aplikace NAT DPDK, která umožňuje zpracovat až 8,5 miliónu paketů za sekundu, což je více, než trojnásobný počet zpracovaných paketů v OS Linux.

Výsledkem této práce je rozšíření aplikace NAT DPDK. V rámci tohoto rozšíření je implementováno rozhraní, umožňující přidávání modulů, které jsou schopny manipulovat s aplikačními daty paketů. To je vhodné pro provádění překladu síťových adres, které jsou uloženy v aplikačních datech. Součástí této práce je implementace tří modulů, které provádějí překlad adres v protokolech SIP, FTP a IRC DCC. Nakonec bylo provedeno měření, jak velký dopad má přidané rozhraní na výkonnost NAT DPDK aplikace. Měření ukázalo, že při použití více jader pro zpracování příchozích rámců je výkonnostní dopad zanedbatelný.

Aktuálně modul, který zpracovává protokol SIP, podporuje pouze příchozí VoIP hovory. Modul jsem však vytvořil tak, aby jej v budoucnosti bylo možné jednoduše rozšířit o funkcionalitu SIP registrátora a SIP proxy. Po takovém rozšíření bude možné v NAT DPDK zpracovávat a směrovat i příchozí VoIP hovory.

Literatura

[1] CHARLES E. SPURGEON a JOANN ZIMMERMAN. *Ethernet: The Definitive Guide*. 2. Aufl. Sebastopol, CA: O'Reilly, 2014. ISBN 14-493-6184-6.

[2] PICCARD, Paul a Marcus SACHS. *Securing IM and P2P applications for the enterprise*. Online-Ausg. Rockland, MA: Syngress Pub, 2006. ISBN 15-974-9017-2.

[3] SANDER VAN VUGT. *Pro Ubuntu server administration*. Berkeley, CA: Apress, 2009. ISBN 978-143-0216-230.

[4] SEIFERT, Rich. *Gigabit Ethernet: technology and applications for high speed LANs*. Addison-Wesley, 1998. ISBN 0201185539.

[5] STEWART, James Michael. *Network security, firewalls, and VPNs*. Second edition. ISBN 12-840-3167-5.

[6] VRÁNA, Roman. *Testování vysokorychlostního nástroje pro překlad IP adres*. Brno, 2016. Diplomová práce. FIT VUT. Vedoucí práce Ing. Martin Žádník, Ph.D.

[7] ALLMAN, M. a W. STEVENS. RFC 2581. *TCP Congestion Control*. IETF, 1999. Dostupné z: https://tools.ietf.org/html/rfc2581

[8] Environment Abstraction Layer. *DPDK* [online] [cit. 2016-05-05]. Dostupné z: http://dpdk.org/doc/guides/prog_guide/env_abstraction_layer.html

[9] HANDLEY, M., H. SCHULZRINNE, E. SCHOOLER a J. ROSENBERG. RFC 2543. *SIP: Session Initiation Protocol*. IETF, 1999. Dostupné také z: https://tools.ietf.org/html/rfc2543

[10] HANDLEY, M., V. JACOBSON a C. PERKINS. RFC 4566. *SDP: Session Description Protocol*. IETF, 2006. Dostupné také z: https://tools.ietf.org/html/rfc4566

[11] Hash Library. *DPDK* [online]. [cit. 2016-05-13]. Dostupné z: http://dpdk.org/doc/guides/prog_guide/hash_lib.html

[12] HOLDREGE, M. a P. SRISURESH. RFC 3027. *Protocol Complications with the IP Network Address Translator*. IETF, 2001. Dostupné také z: https://tools.ietf.org/html/rfc3027

[13] Intel Co. *Intel® 64 and IA-32 Architectures Developer's Manual: Volume 3A: System Programming Guide, Part 1* [online]. 2016 [cit. 2016-05-15]. Dostupné z: http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html

[14] Mbuf Library. *DPDK* [online]. [cit. 2016-05-13]. Dostupné z: http://dpdk.org/doc/guides/prog_guide/mbuf_lib.html

[15] Mempool Library. *DPDK* [online]. [cit. 2016-05-13]. Dostupné z: http://dpdk.org/doc/guides/prog_guide/mempool_lib.html

[16] NAPI. *Linux Foundation* [online]. 2009 [cit. 2016-01-05]. Dostupné z: http://www.linuxfoundation.org/collaborate/workgroups/networking/napi

[17] OIKARINEN, J. a D. REED. RFC 1495. *Internet Relay Chat Protocol*. IETF, 1993. Dostupné také z: https://tools.ietf.org/html/rfc1459

[18] Poll Mode Driver. *DPDK* [online]. [cit. 2016-05-13]. Dostupné z: http://dpdk.org/doc/guides/prog_guide/poll_mode_drv.html

[19] POSTEL, J. a J. REYNOLDS. RFC 959. *FILE TRANSFER PROTOCOL (FTP)*. 1985. Dostupné také z: https://tools.ietf.org/html/rfc959

[20] QUITTEK, J., T. ZSEBY, B. CLAISE a S. ZANDER. RFC 3917. *Requirements for IP Flow Information Export (IPFIX)*. IETF, 2004. Dostupné z: https://tools.ietf.org/html/rfc3917

[21] RESNICK, P. QUALCOMM INCORPORATED. RFC 2822 *Internet Message Format*. IETF, 2001. Dostupné také z: https://tools.ietf.org/html/rfc2822

[22] Ring Library. *DPDK* [online]. [cit. 2016-05-13]. Dostupné z: http://dpdk.org/doc/guides/prog_guide/ring_lib.html

[23] ROSENBERG, J., H. SCHULZRINNE, G. CAMARILLO, A. JOHNSTON, J. PETERSON, R. SPARKS, M. HANDLEY a E. SCHOOLER. RFC 3261. *SIP: Session Initiation Protocol*. 2002. Dostupné také z: https://tools.ietf.org/html/rfc3261

[24] SENIE, D. AMARANTH NETWORKS INC. RFC3235. *Network Address Translator (NAT)-Friendly: Application Design Guidelines*. IETF, 2002. Dostupné také z: https://tools.ietf.org/html/rfc3235

[25] SRISURESH, P. a D. GAN. RFC 2391. *Load Sharing using IP Network Address Translation (LSNAT)*. IETF, 1998. Dostupné z: https://tools.ietf.org/html/rfc2391

[26] SRISURESH, P. a K. EGEVANG. RFC 3022. *Traditional IP Network Address Translator (Traditional NAT)*. IETF, 2001. Dostupné z: https://www.tools.ietf.org/html/rfc3022

[27] SRISURESH, P. a M. HOLDREGE. RFC 2663. *IP Network Address Translator (NAT) Terminology and Considerations*. IETF, 1999. Dostupné z: https://tools.ietf.org/html/rfc2663

[28] UNIVERSITY OF SOUTHERN CALIFORNIA. RFC 793. *TRANSMISSION CONTROL PROTOCOL: DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION*. IETF, 1981. Dostupné z: https://tools.ietf.org/html/rfc793

Přílohy

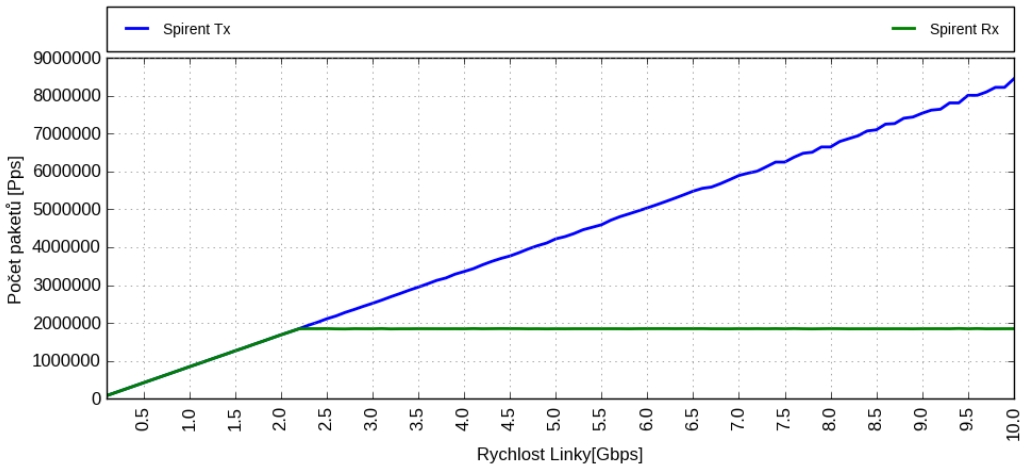
Seznam příloh

A Výsledky všech provedených měření **60**
A.1 AMD Opteron 6376 10Gbps – Netfilter 60
A.2 Intel Xeon E5-2620 10Gbps – Netfilter 64
A.3 Intel i3-6300 1Gbps – Netfilter 68
A.4 Intel i7-6700 1Gbps – Netfilter 71
A.5 Intel Xeon E5-2620 10Gbps – NAT DPDK 72

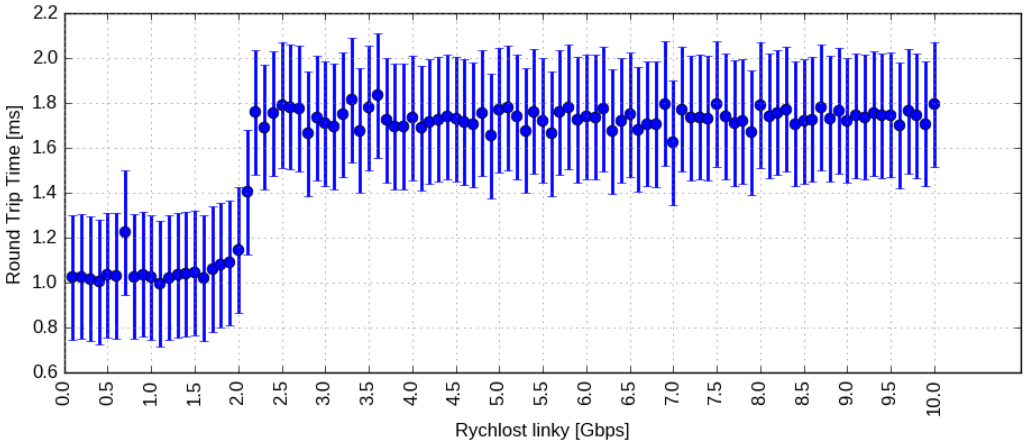
Příloha A

Výsledky všech provedených měření

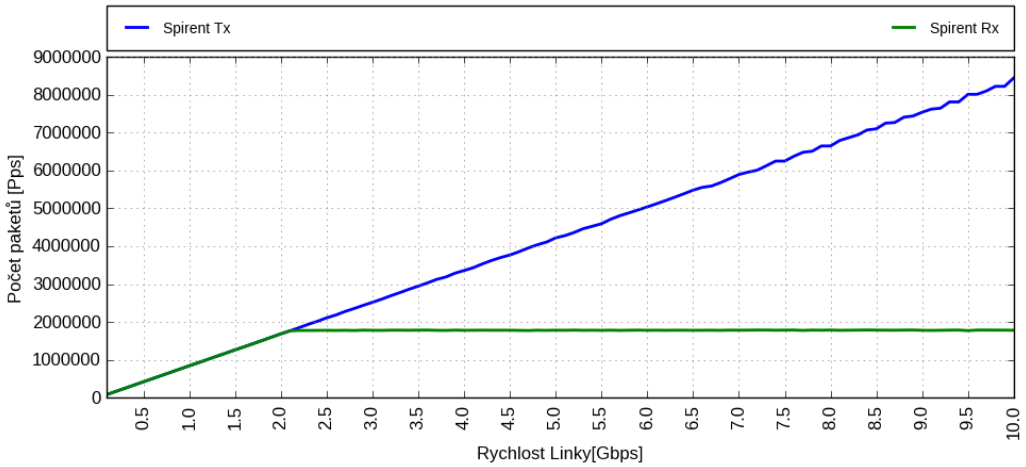
A.1 AMD Opteron 6376 10Gbps – Netfilter



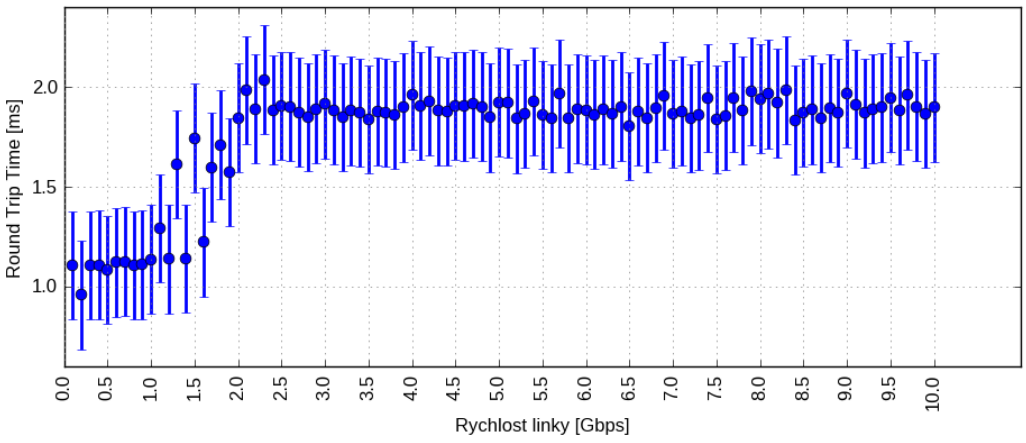
Obrázek A.1: AMD Opteron – graf závislosti propustnosti na zátěži linky, 1000 pravidel



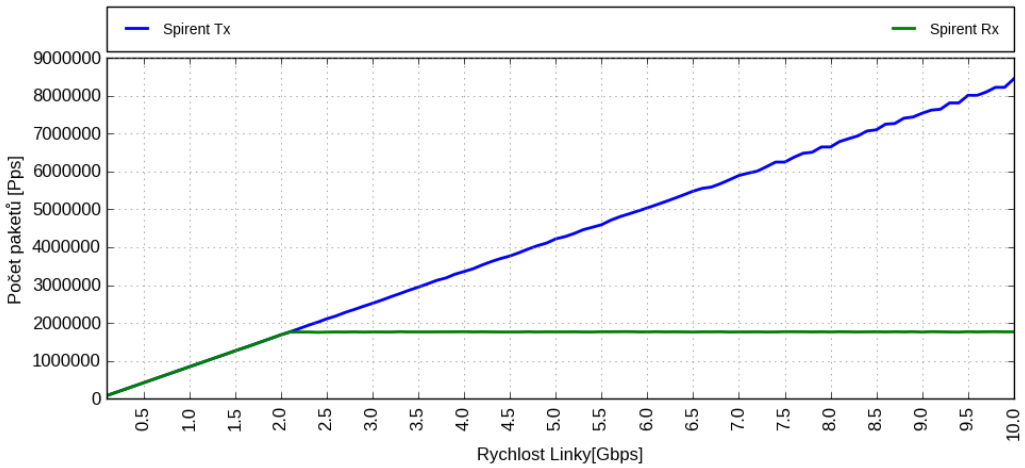
Obrázek A.2: AMD Opteron – graf závislosti RTT na zátěži linky, 1000 pravidel



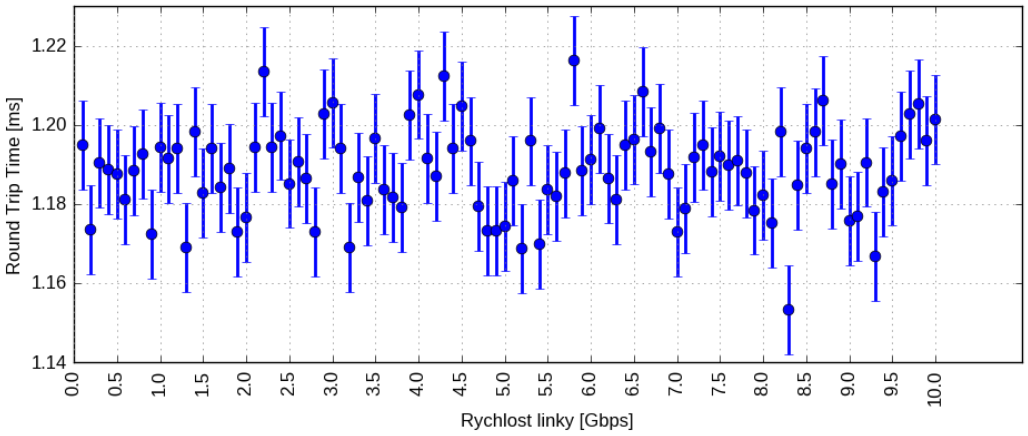
Obrázek A.3: AMD Opteron – graf závislosti propustnosti na zátěži linky, 3000 pravidel



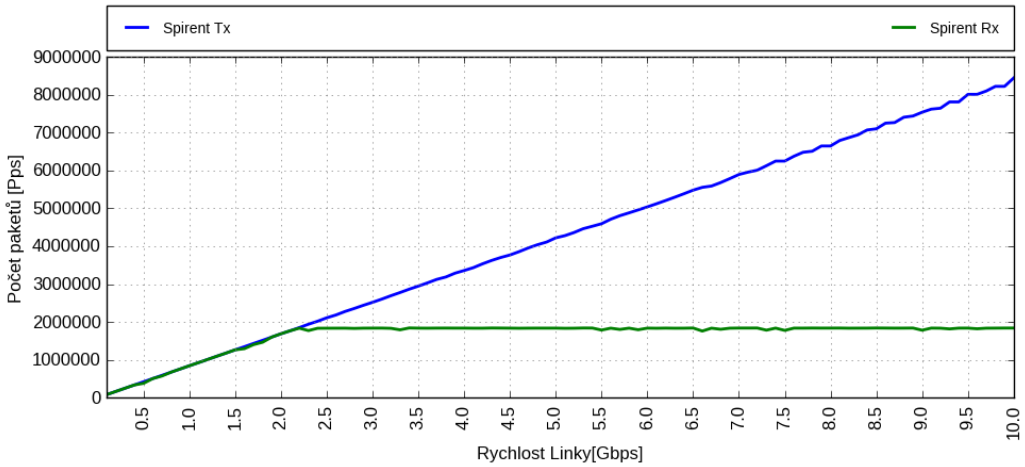
Obrázek A.4: AMD Opteron – graf závislosti RTT na zátěži linky, 3000 pravidel



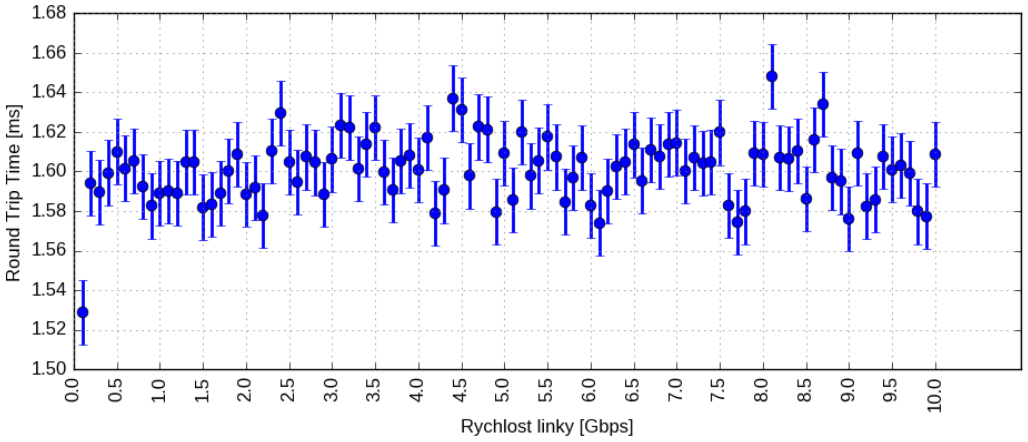
Obrázek A.5: AMD Opteron – graf závislosti propustnosti na zátěži linky, 5000 pravidel



Obrázek A.6: AMD Opteron – graf závislosti RTT na zátěži linky, 5000 pravidel

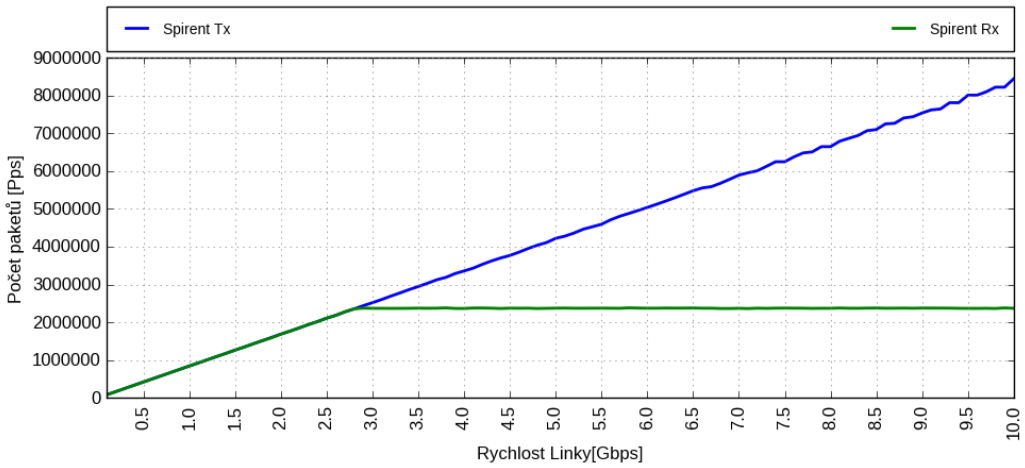


Obrázek A.7: AMD Opteron – graf závislosti propustnosti na zátěži linky, 10000 pravidel

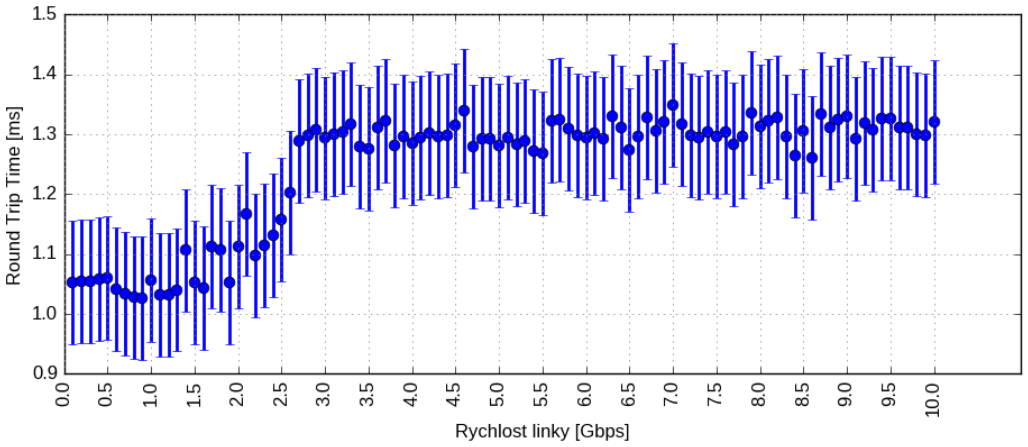


Obrázek A.8: AMD Opteron – graf závislosti RTT na zátěži linky, 10000 pravidel

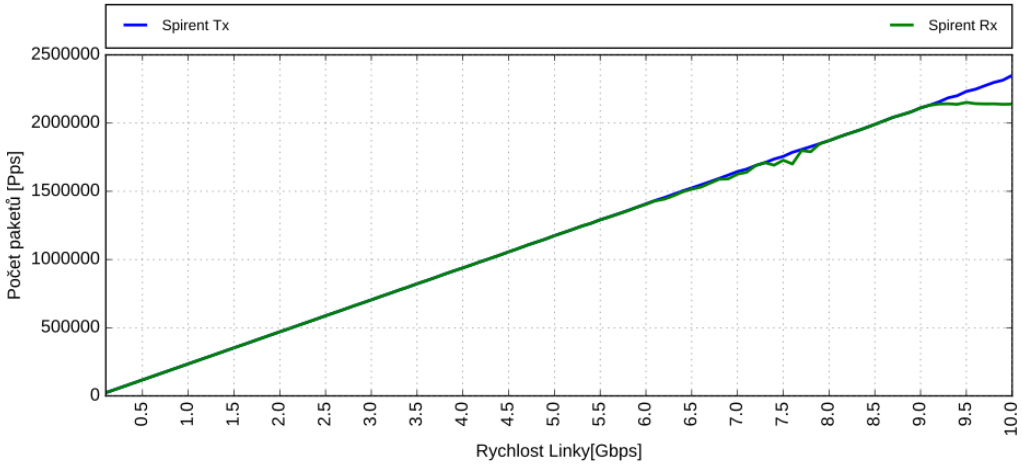
A.2 Intel Xeon E5-2620 10Gbps – Netfilter



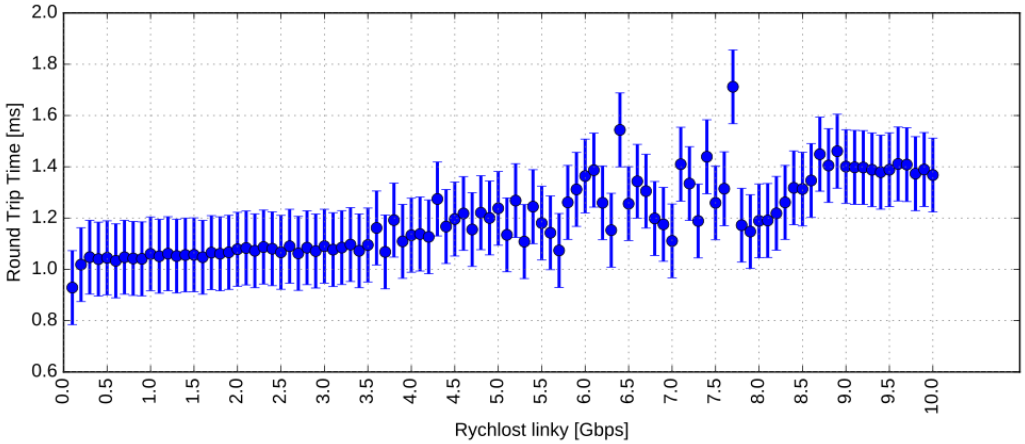
Obrázek A.9: Intel Xeon – graf závislosti propustnosti na zátěži linky, 1000 pravidel



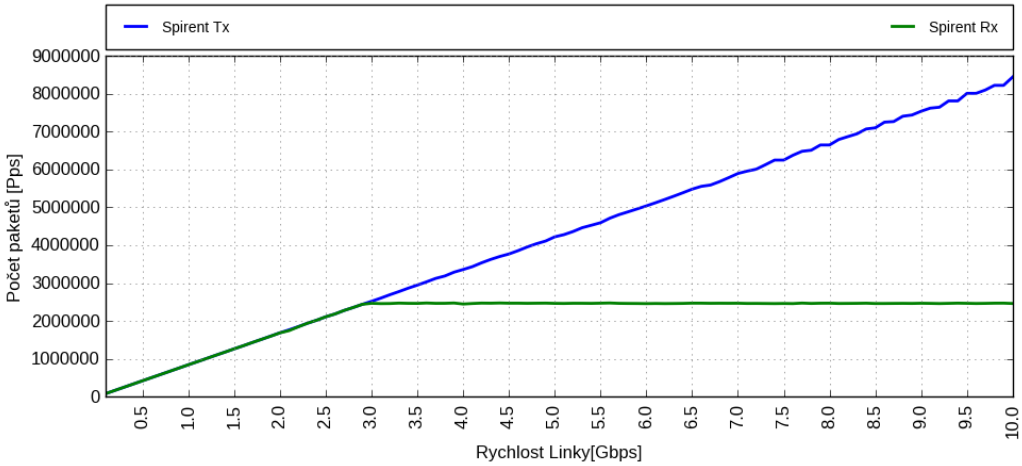
Obrázek A.10: Intel Xeon – graf závislosti RTT na zátěži linky, 1000 pravidel



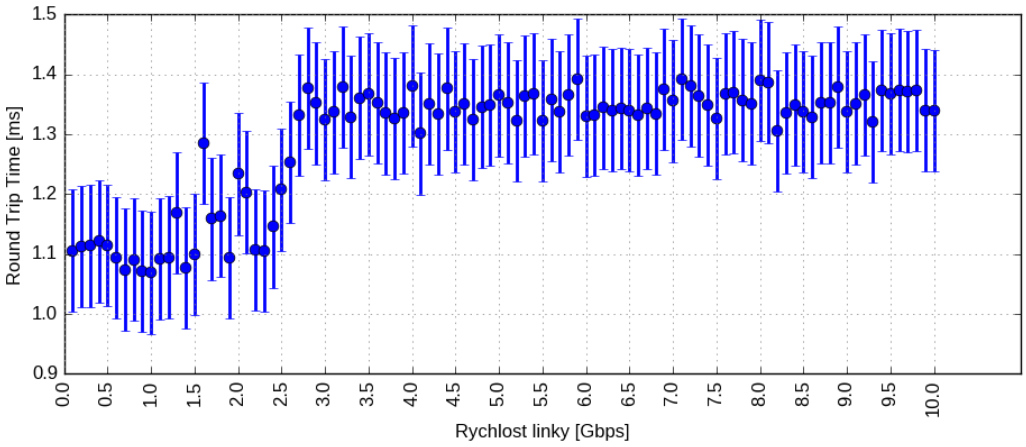
Obrázek A.11: Intel Xeon – graf závislosti propustnosti na zátěži linky, 1000 pravidel, 512B pakety



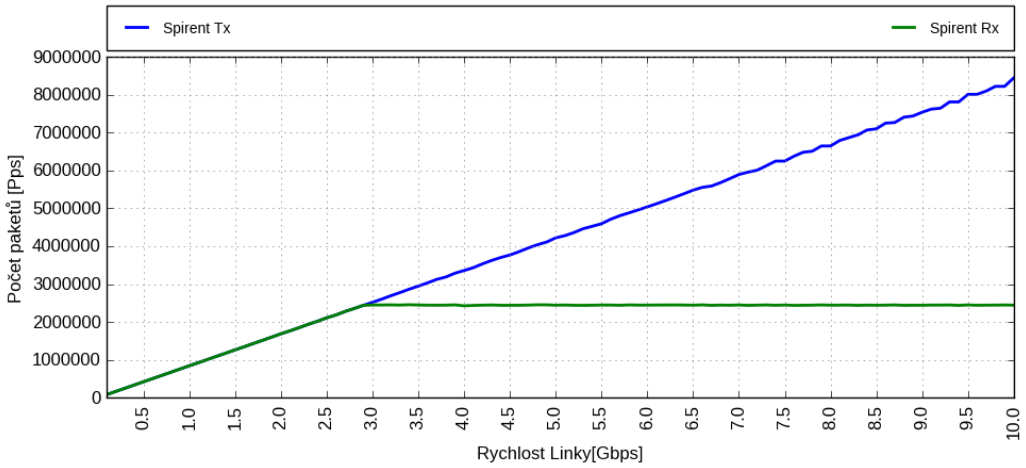
Obrázek A.12: Intel Xeon – graf závislosti RTT na zátěži linky, 1000 pravidel, 512B pakety



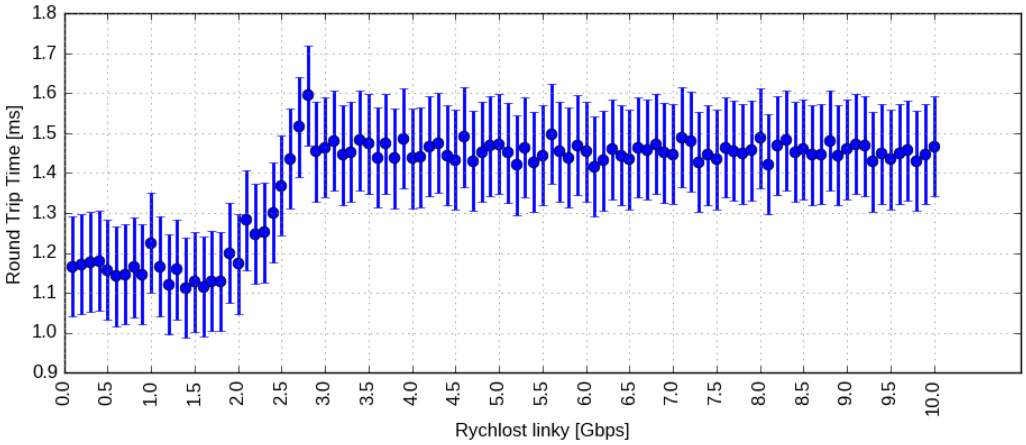
Obrázek A.13: Intel Xeon – graf závislosti propustnosti na zátěži linky, 3000 pravidel



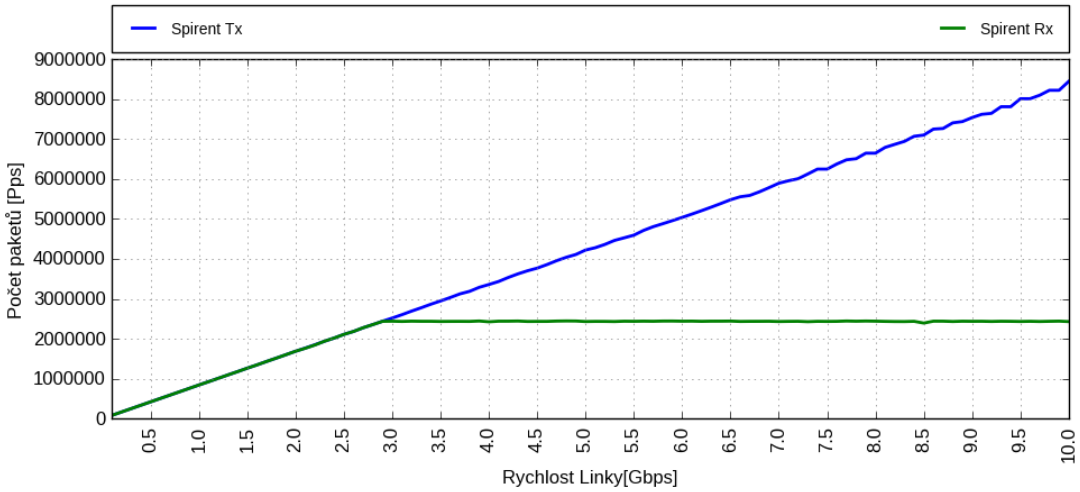
Obrázek A.14: Intel Xeon – graf závislosti RTT na zátěži linky, 3000 pravidel



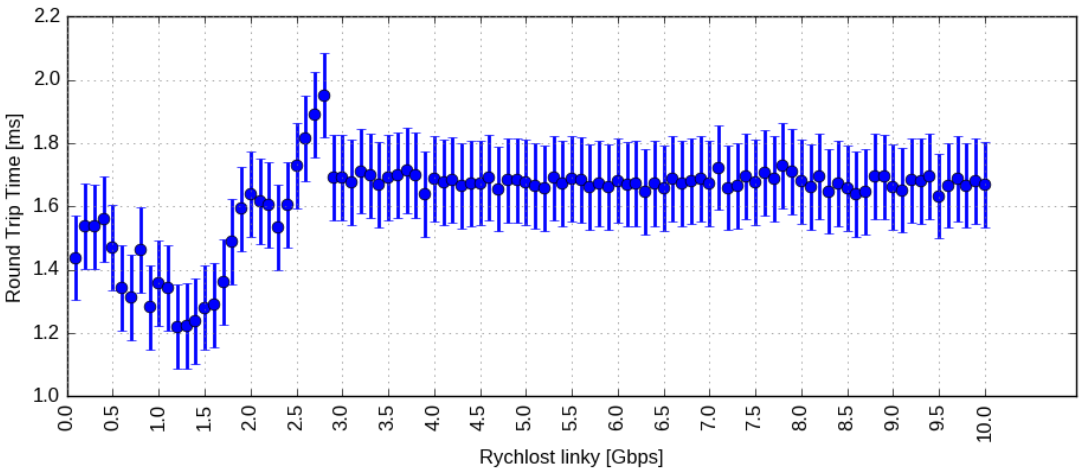
Obrázek A.15: Intel Xeon – graf závislosti propustnosti na zátěži linky, 5000 pravidel



Obrázek A.16: Intel Xeon – graf závislosti RTT na zátěži linky, 5000 pravidel

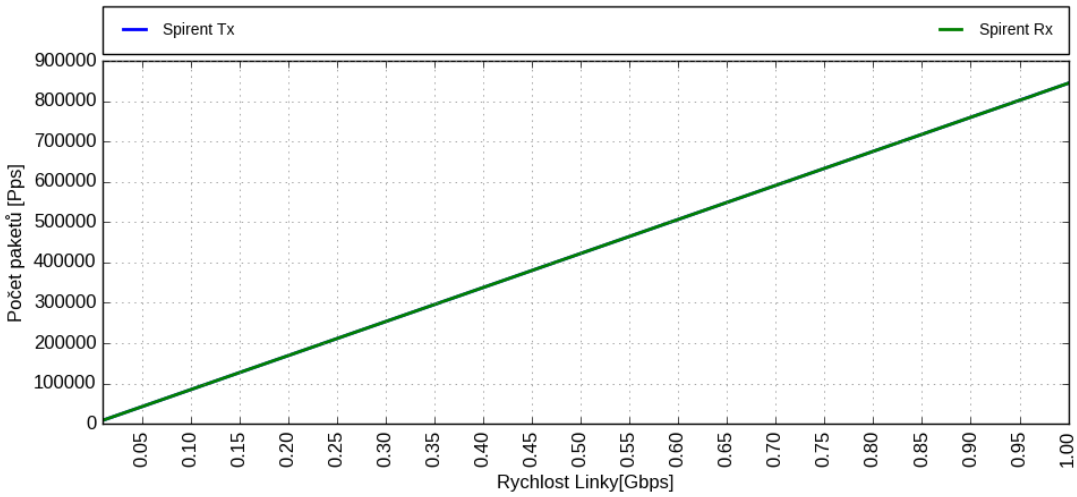


Obrázek A.17: Intel Xeon – graf závislosti propustnosti na zátěži linky, 10000 pravidel

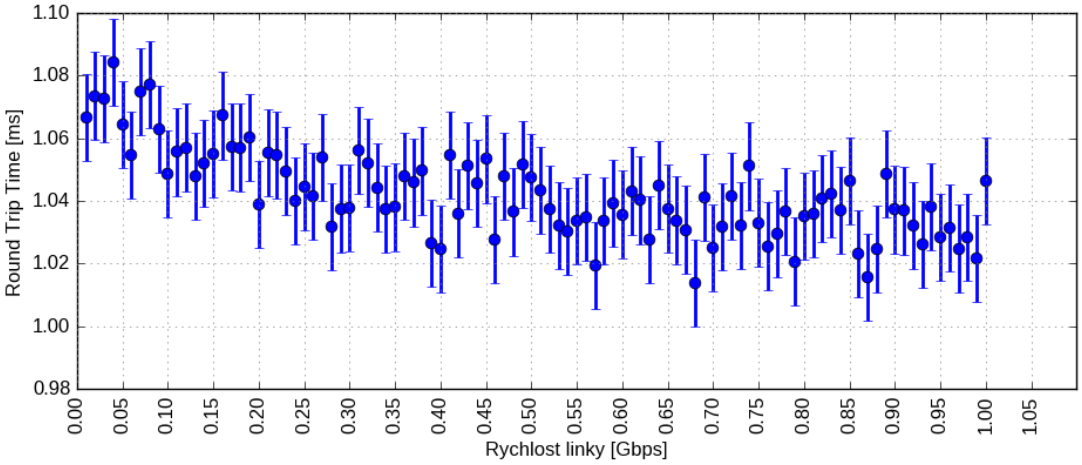


Obrázek A.18: Intel Xeon – graf závislosti RTT na zátěži linky, 10000 pravidel

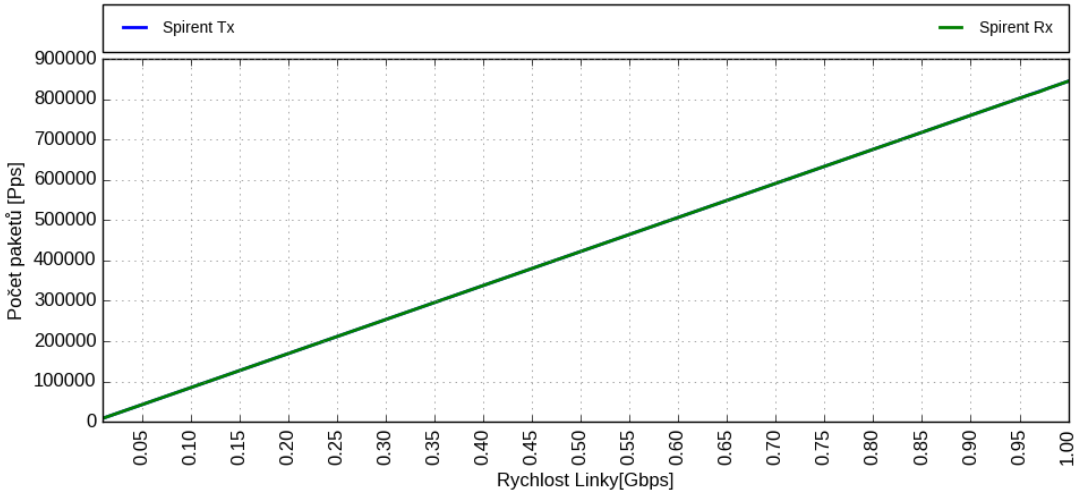
A.3 Intel i3-6300 1Gbps – Netfilter



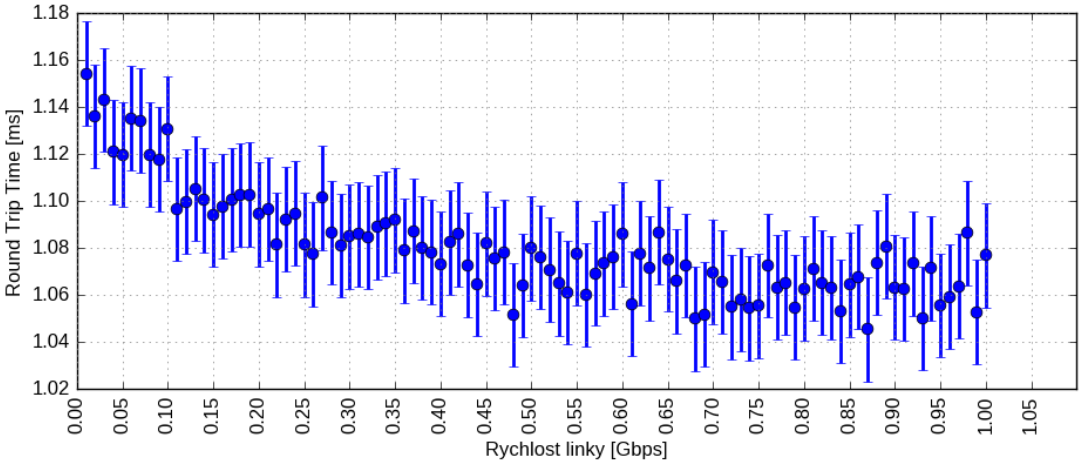
Obrázek A.19: Intel i3 – graf závislosti propustnosti na zátěži linky, 1000 pravidel



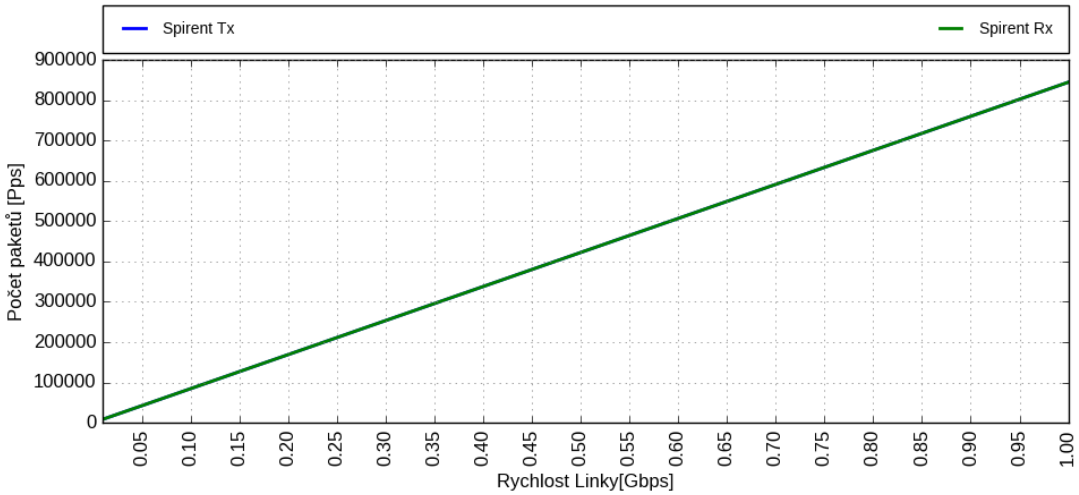
Obrázek A.20: Intel i3 – graf závislosti RTT na zátěži linky, 1000 pravidel



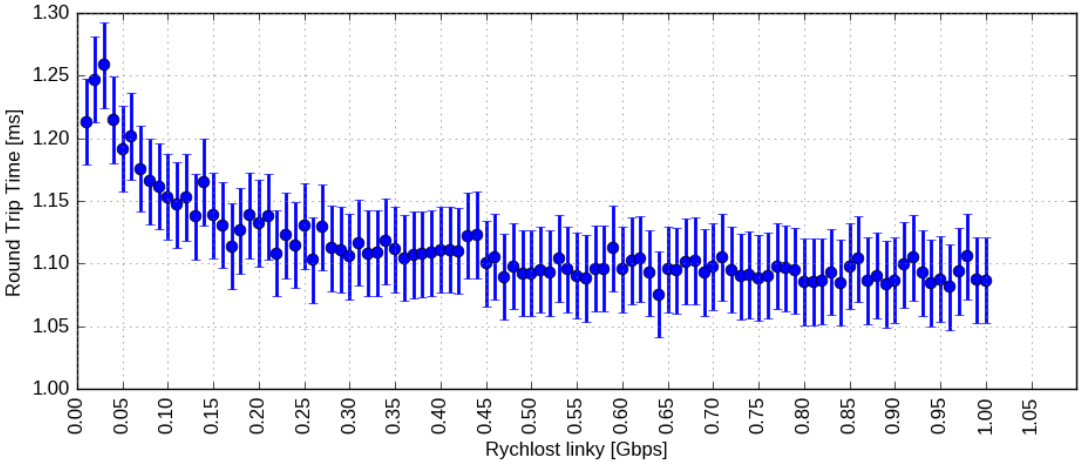
Obrázek A.21: Intel i3 – graf závislosti propustnosti na zátěži linky, 3000 pravidel



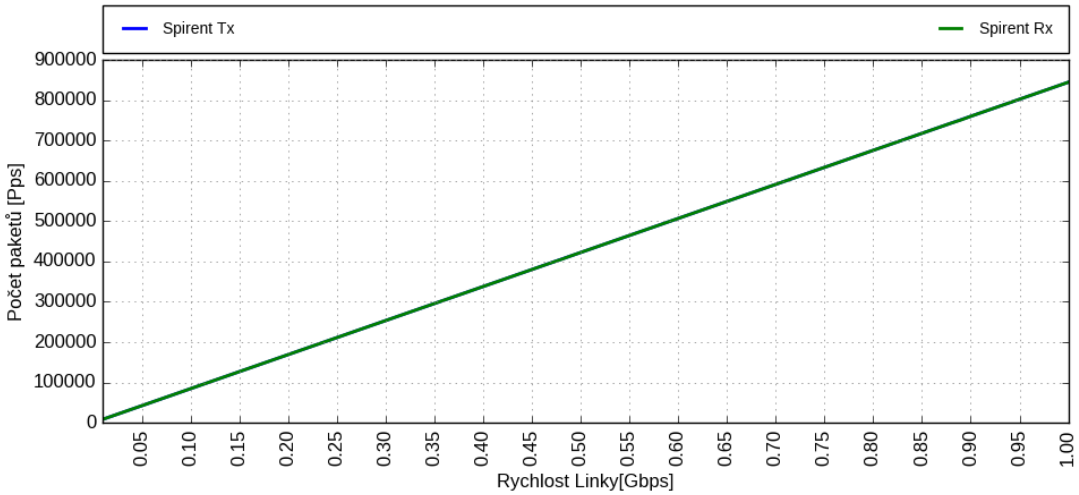
Obrázek A.22: Intel i3 – graf závislosti RTT na zátěži linky, 3000 pravidel



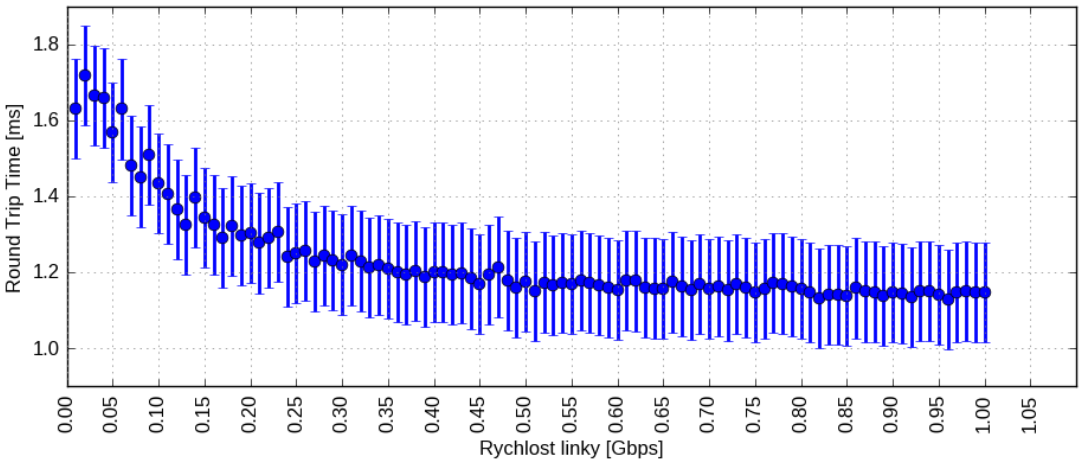
Obrázek A.23: Intel i3 – graf závislosti propustnosti na zátěži linky, 5000 pravidel



Obrázek A.24: Intel i3 – graf závislosti RTT na zátěži linky, 5000 pravidel

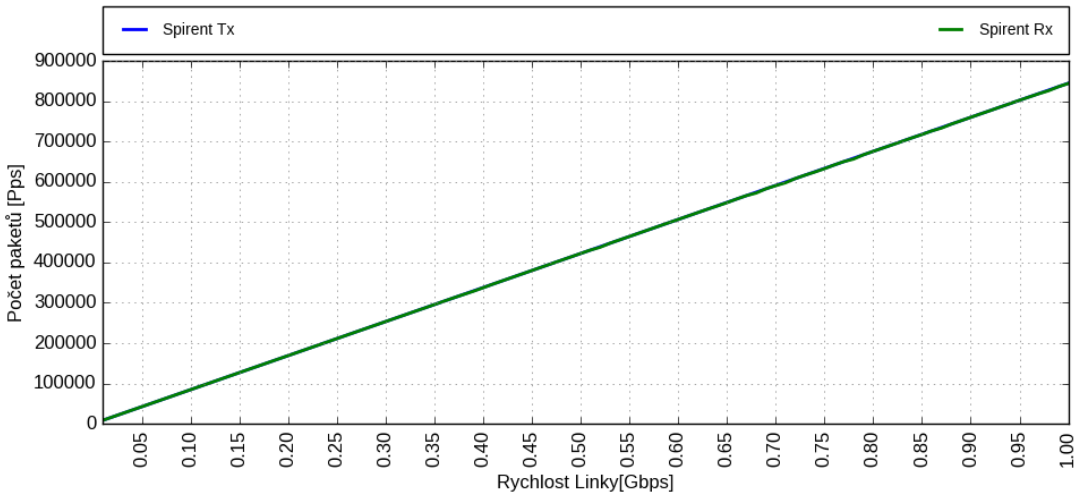


Obrázek A.25: Intel i3 – graf závislosti propustnosti na zátěži linky, 10000 pravidel

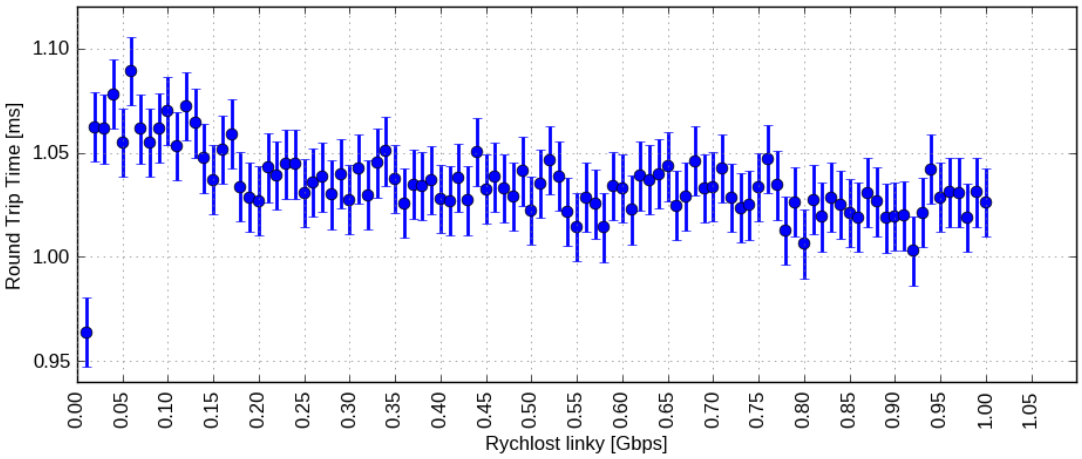


Obrázek A.26: Intel i3 – graf závislosti RTT na zátěži linky, 10000 pravidel

A.4 Intel i7-6700 1Gbps – Netfilter

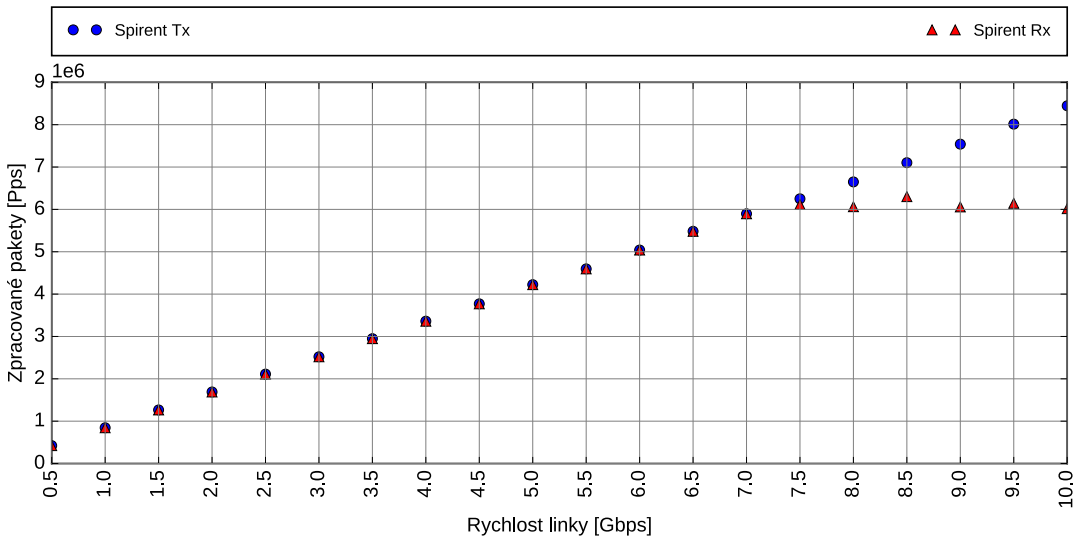


Obrázek A.27: Intel i7 – graf závislosti propustnosti na zátěži linky, 1000 pravidel

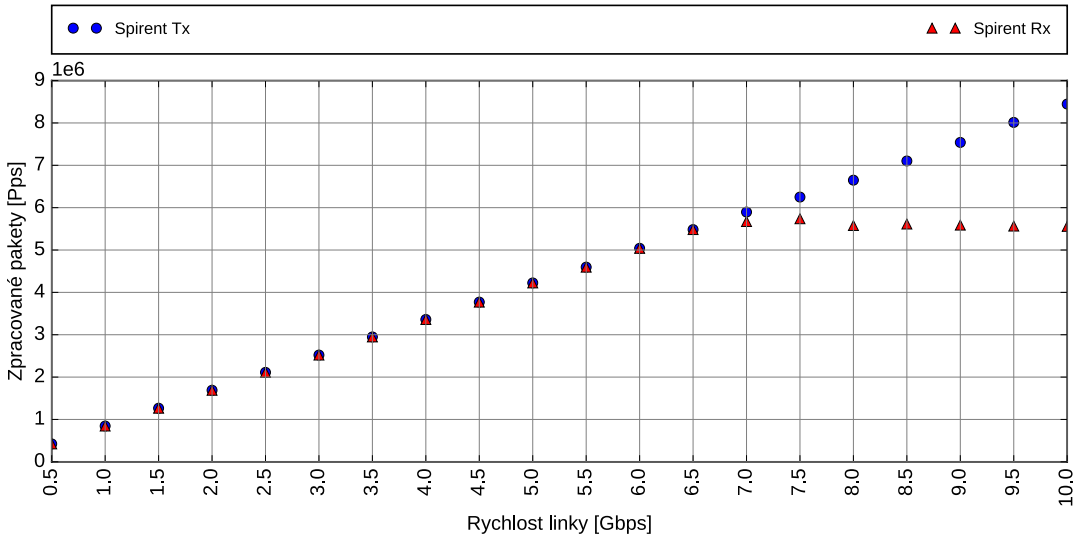


Obrázek A.28: Intel i7 – graf závislosti RTT na zátěži linky, 1000 pravidel

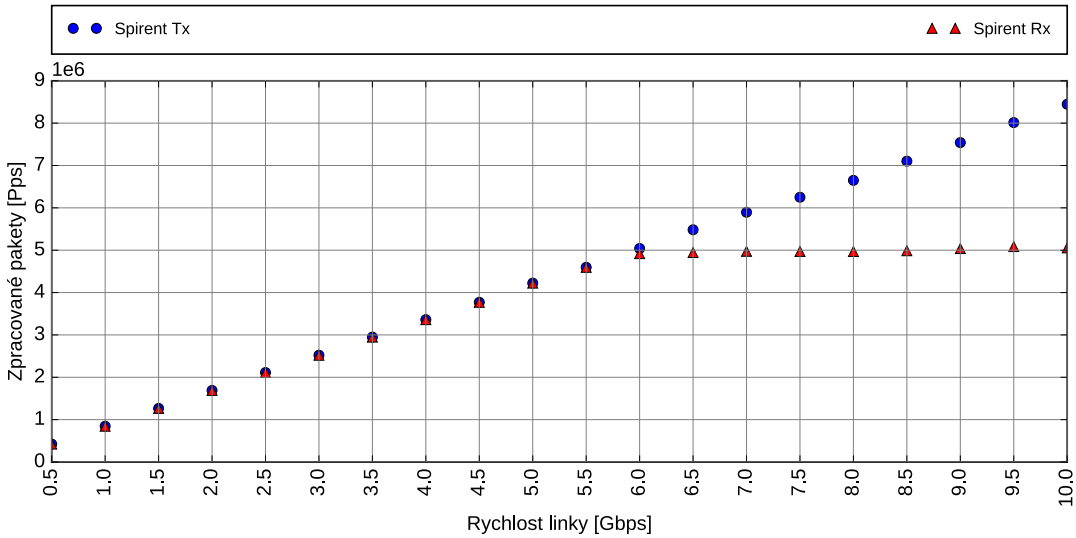
A.5 Intel Xeon E5-2620 10Gbps – NAT DPDK



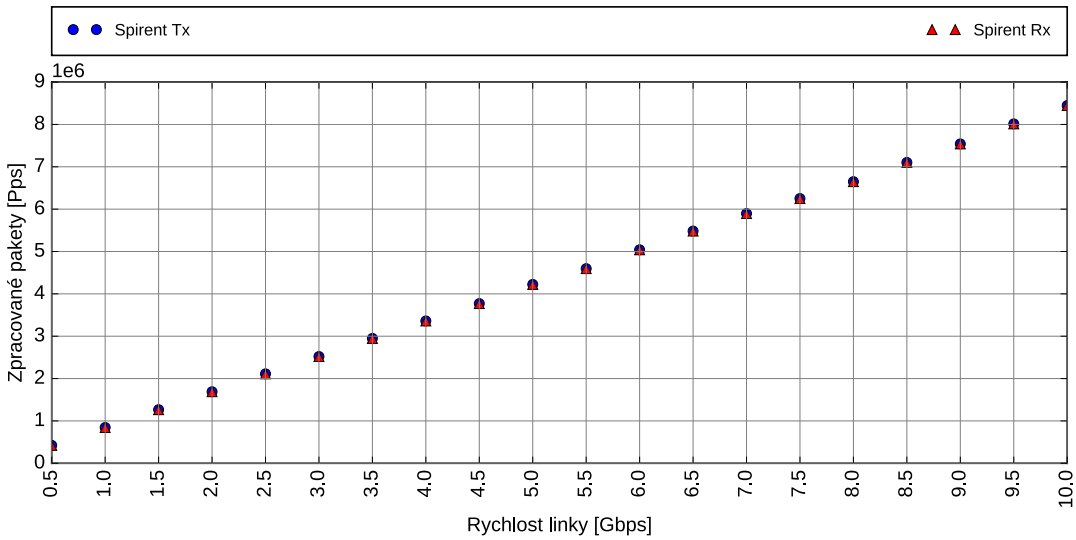
Obrázek A.29: Intel Xeon – graf závislosti propustnosti na zátěži linky bez ALG, 128 B, 1 jádro



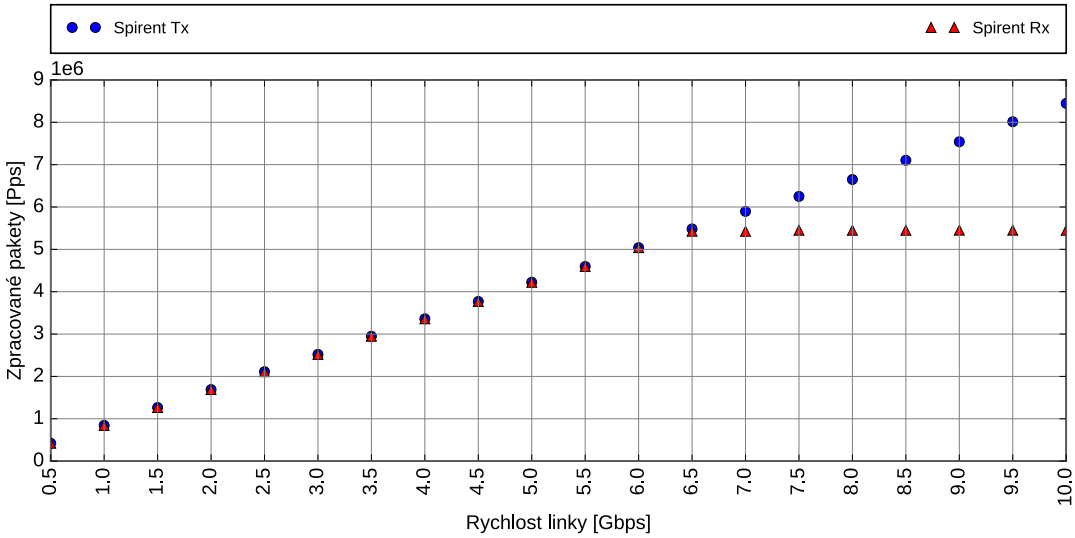
Obrázek A.30: Intel Xeon – graf závislosti propustnosti na zátěži linky s ALG, 128 B, 1 jádro



Obrázek A.31: Intel Xeon – závislosti propustnosti na zátěži linky s ALG, 128 B, 2 jádra



Obrázek A.32: Intel Xeon – graf závislosti propustnosti na zátěži linky, 128 B, 3 jádra



Obrázek A.33: Intel Xeon – graf závislosti propustnosti na zátěži linky, 128 B, 12 jader