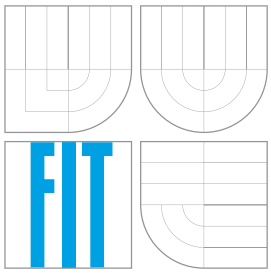
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

TESTOVÁNÍ VYSOKORYCHLOSTNÍHO NÁSTROJE PRO PŘEKLAD IP ADRES

TESTING OF HIGH-SPEED TOOL FOR NETWORK ADDRESS TRANSLATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ROMAN VRÁNA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MARTIN ŽÁDNÍK, Ph.D.

BRNO 2016

Zadání diplomové práce/18594/2015/xvrana20

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Vrána Roman, Bc.**
Obor: Počítačové sítě a komunikace
Téma: **Testování vysokorychlostního nástroje pro překlad IP adres**
Testing of high-speed tool for network address translation
Kategorie: Počítačové sítě

Pokyny:

1. Nastudujte možnosti konfigurace nástroje pro překlad adres (NAT). Nastudujte možnosti testování funkcionality a výkonnosti NAT.
2. V síťové laboratoři připravte testovací prostředí.
3. Navrhněte řešení pro testování nástroje pro překlad adres.
4. Navržené řešení implementujte.
5. Implementované řešení využijte pro testování implementací NAT. Výsledky testů důkladně popište.
6. Zhodnoťte dosažené výsledky a diskutujte možná rozšíření.

Literatura:

- Dle pokynů vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 až 3 zadání.

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

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

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

Vedoucí: **Žádník Martin, Ing., Ph.D.**, UPSY 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

Tato práce prezentuje problematiku testování nástrojů pro překlad adres. Popisuje základy překladu IP adres, obecné metody testování a jejich aplikaci na překladové nástroje. Dále prezentuje návrh a implementaci aplikace pro provádění testovacích metod uvedených v práci. Nástroj je pak použit pro testování nově vyvíjeného vysokorychlostního nástroje pro překlad adres.

Abstract

This master thesis presents a topic of testing network address translation. It describes basic principles of network address translation and general methods for testing network devices. These methods are then applied on NAT devices. Thesis then introduces a design of a framework for performing functional tests. The designed framework is then implemented as a part of the thesis and used to test a newly developed high-speed NAT application.

Klíčová slova

Překlad IP adres, testování, vysokorychlostní přenos dat

Keywords

Network address translation, testing, high speed data transfer

Citace

Roman Vrána: Testování vysokorychlostního nástroje
pro překlad IP adres, diplomová práce, Brno, FIT VUT v Brně, 2016

Testování vysokorychlostního nástroje pro překlad IP adres

Prohlášení

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

.....
Roman Vrána
23. května 2016

Poděkování

Rád bych poděkoval Ing. Martinu Žádníkovi, Ph.D. za vedení diplomové práce. Dále bych rád poděkoval vývojovému týmu projektu SProbe za poskytnutí vývojových a testovacích prostředků

Obsah

Kapitola 1

Úvod

Síťová komunikace je v současnosti provozována převážně na protokolu IPv4. Tento protokol adresuje zařízení na logické úrovni a je využíván ke směrování dat. Pokud však zanedbáme některé vyhrazené rozsahy, zjistíme, že počet dostupných adres nedokáže pokrýt všechna zařízení. Protokol IPv6, jenž by měl nedostatek síťových adres řešit, však stále není dostatečně rozšířený. Musíme tedy použít takové řešení, které by nám umožnilo docílit požadované konektivity a zároveň snížilo počet potřebných adres. Nejběžnějším řešením je použití překladu síťových adres neboli NAT. S jeho pomocí můžeme například sdílet jednu adresu směrovatelnou v síti internetu mezi více stanicemi. S rostoucími nároky na rychlost a kvalitu přenosu je také nutné vyvíjet nové nástroje, které by uspokojily stanovené požadavky. U těchto nástrojů je pak nutné ověřit, zda provádějí překlad adres správným způsobem. To znamená, že upravují správné pakety a nepoškozují data či jinak nenarušují komunikaci. Dále je také nutné ověřit, jak jsou takové nástroje výkonné, tedy jestli mají dostatečnou či avizovanou propustnost, nebo jestli nemají negativní dopady na odezvu. Tato práce se bude zabývat problematikou ověřování těchto parametrů, návrhem a implementací nástroje, jenž by tyto testy umožnil a automatizoval. Činnost tohoto nástroje pak bude demonstrována na vybraných testovacích scénářích.

V úvodu práce bude stručně nastíněna obecná problematika překladu a její aplikace na současná řešení. Dále budou rozebrány obecné způsoby testování síťových zařízení a jejich aplikace na nástroje pro překlad adres. V kapitole 6 pak bude popsán návrh testovacího nástroje, uvedený v předcházejícím semestrálním projektu, a jeho implementace. Bude specifikována struktura z hlediska jednotlivých modulů a jejich role v probíhajících testech. Dále budou uvedeny testovací případy použité pro ověření činnosti nástroje pro překlad. Ty budou pak aplikovány na současné implementace nástrojů pro překlad adres. K závěru pak budou diskutována možná rozšíření testovacího nástroje o další metody testů, či podrobnější způsoby vyhodnocování výsledků.

Kapitola 2

Překlad síťových adres v praxi

Před popisem návrhu testovacího prostředí je vhodné nejprve popsat problematiku překladu adres. V této kapitole bude tedy rozebrána funkce překladu síťové adresy pro přenos dat. Budou popsány způsoby jak statického, tak dynamického překladu s použitím nastaveného rozsahu adres. Kapitola čerpá informace převážně z příslušných dokumentů RFC a dále z článku Anatomy: A Look Inside Network Address Translators [11].

2.1 Základy překladu síťových adres

Funkce překladu adres (Network Address Translation podle RFC 3022 [18], dále jen NAT) je založena na modifikaci identifikátorů síťové vrstvy, konkrétněji adresy protokolu IPv4 [13]. Ty jsou používány při směrování dat v síti. Všechny adresy, které jsou použity v síti internetu, spadají do tzv. veřejných adres. Ty jsou přidělovány na základě registrace dané entity u regionálních registrátorů (RIPE NCC[1] pro Evropu nebo ARIN[2] pro Spojené Státy a Kanadu). Mimo takto přidělené adresy existují rozsahy adres vyhrazené k dalším účelům. Jedná se například o adresy pro multicast, adresy označující loopback rozhraní nebo tzv. privátní rozsahy adres. Všechny tyto adresy nelze v síti internetu směrovat a pokud se vyskytne paket s takovou adresou, měly by jej směrovače zahodit.

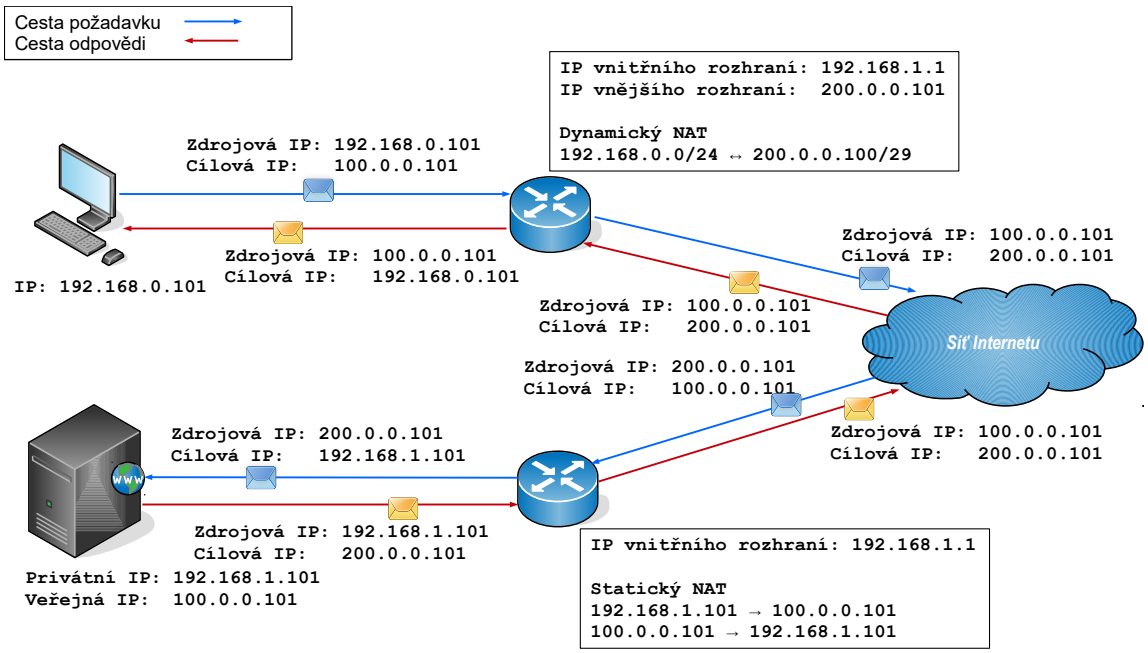
Poslední jmenovaný rozsah je velmi často spojen právě s překladem adres. Privátní adresy jsou totiž běžně přidělovány zařízením, jež se nacházejí např. v jedné budově nebo místnosti. Pokud bychom požadovali, aby tato zařízení měla přístup k internetu, museli bychom každému z nich přiřadit veřejnou adresu. V současnosti však toto řešení není možné vzhledem k nedostupnosti dostatečného počtu těchto adresu. Můžeme však přidělit veřejnou adresu výchozí bráně této privátní sítě. Musíme však zajistit, aby pakety opouštějící síť tuto adresu obdržely. To nám zajistí právě překlad adres.

Při překladu adres bude výchozí brána provádět modifikaci zdrojové nebo cílové adresy na základě směru, ze kterého bude provoz přiveden. Pro tyto úpravy využívá vyhrazená pravidla, podle který provádí potřebné modifikace. Přestože primárně je NAT navržen na překlad síťových adres může být nastaven do takové podoby, aby do překladu zahrnoval i identifikátory transportní vrstvy (dále jen porty), případně upravoval i data některých aplikačních protokolů, aby zajistil plnou konektivitu.

[1] Réseaux IP Européens Network Coordination Centre
[2] American Registry for Internet Numbers

2.1.1 Příklady průchodu paketu skrze NAT

Jako příklad běžného užití překladu si můžeme uvést komunikaci stanice umístěné v sítí s privátním adresováním a serveru umístěném v síti internetu. Tuto komunikaci nelze díky struktuře sítě internet provést, protože pakety s privátními adresami není dovoleno směrovat ve veřejné síti. Klient je sice schopen otevřít spojení, ale server nemůže odpovědět. Nastavíme-li na vstupním bodu překlad síťových adres pak je paket při cestě podroben modifikaci podle pravidla pro překlad. Při něm dojde k úpravě zdrojové adresy. Ta je buď dána rozsahem veřejných adres, ve kterém leží výstupní rozhraní brány nebo se použije právě adresa výstupního rozhraní. Upravený paket je pak vyslán do sítě. Cílová stanice zpracuje data stejným způsobem, jakoby je obdržela od klienta, a odešle zpátky. Zařízení, které provedlo překlad pak po obdržení odpovědi modifikuje cílovou adresu zpět na stanici, která vyslala požadavek. Data jsou pak doručena obvyklým způsobem. Průchod dat je ilustrován na obrázku 2.1.

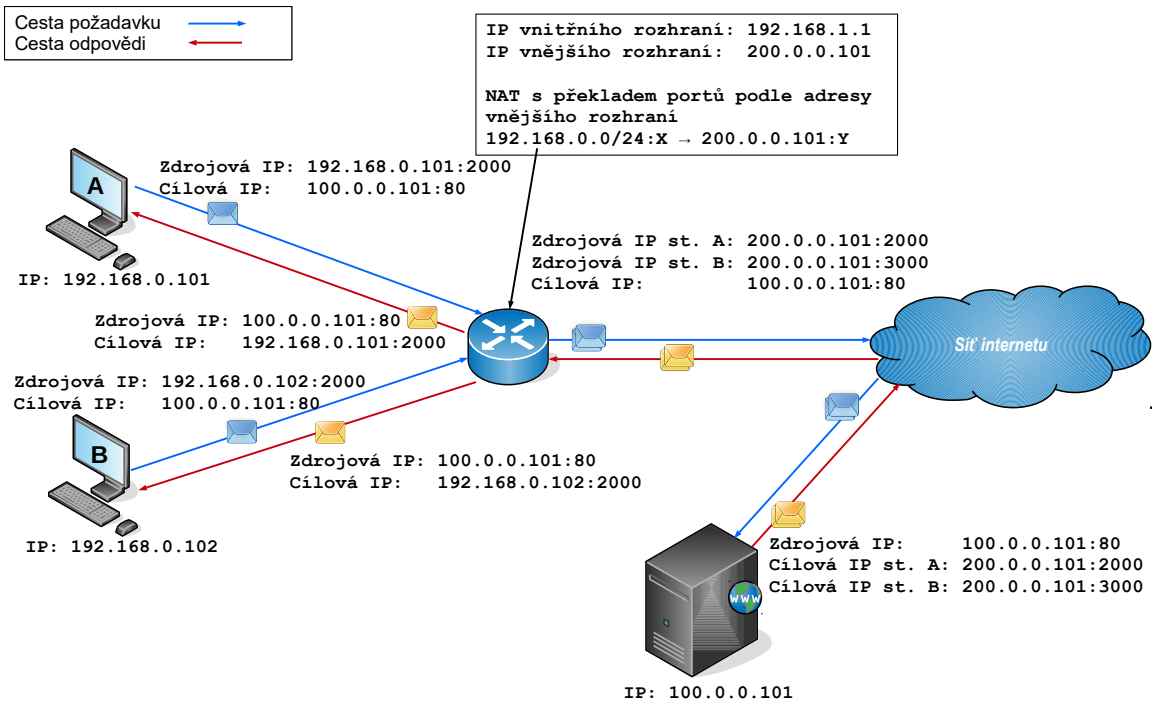


Obrázek 2.1: Průchod paketu skrze NAT

Uvedený způsob umožňuje spojení dvou stanic, kdy jedna z nich je v privátním adresovém prostoru. Pokud by takových stanic bylo více, pak musíme pro každou z nich specifikovat adresu, která bude použita při překladu. Tu lze přiřadit staticky, pak hovoříme o překladu 1:1. Tímto pravidlem je možné zachovat funkcionalitu na úrovni spojení point-to-point. Vyžaduje to však dostupnost dostatečného počtu veřejných adres. Pokud však nevyžadujeme nutně zachování přímého spojení, pak můžeme pouze vyhradit rozsah adres pro překlad a samotný výběr nechat na zařízení. V tomto případě provádíme mapování M:N.

V běžné praxi se také setkáme s použitím NAPT (Port-Translating NAT). Tento způsob překladu provádí nejen úpravu síťových adres, ale také portů. S jeho pomocí je tedy možné otevřít vetší množství spojení s použitím stejné síťové adresy (efektivně 2^{16} spojení). Ta je většinou shodná jako adresa rozhraní připojeného k veřejné síti. Tento způsob však omezuje resp. znemožňuje komunikaci point-to-point. Mapování, byť se v rámci spojení nezmění,

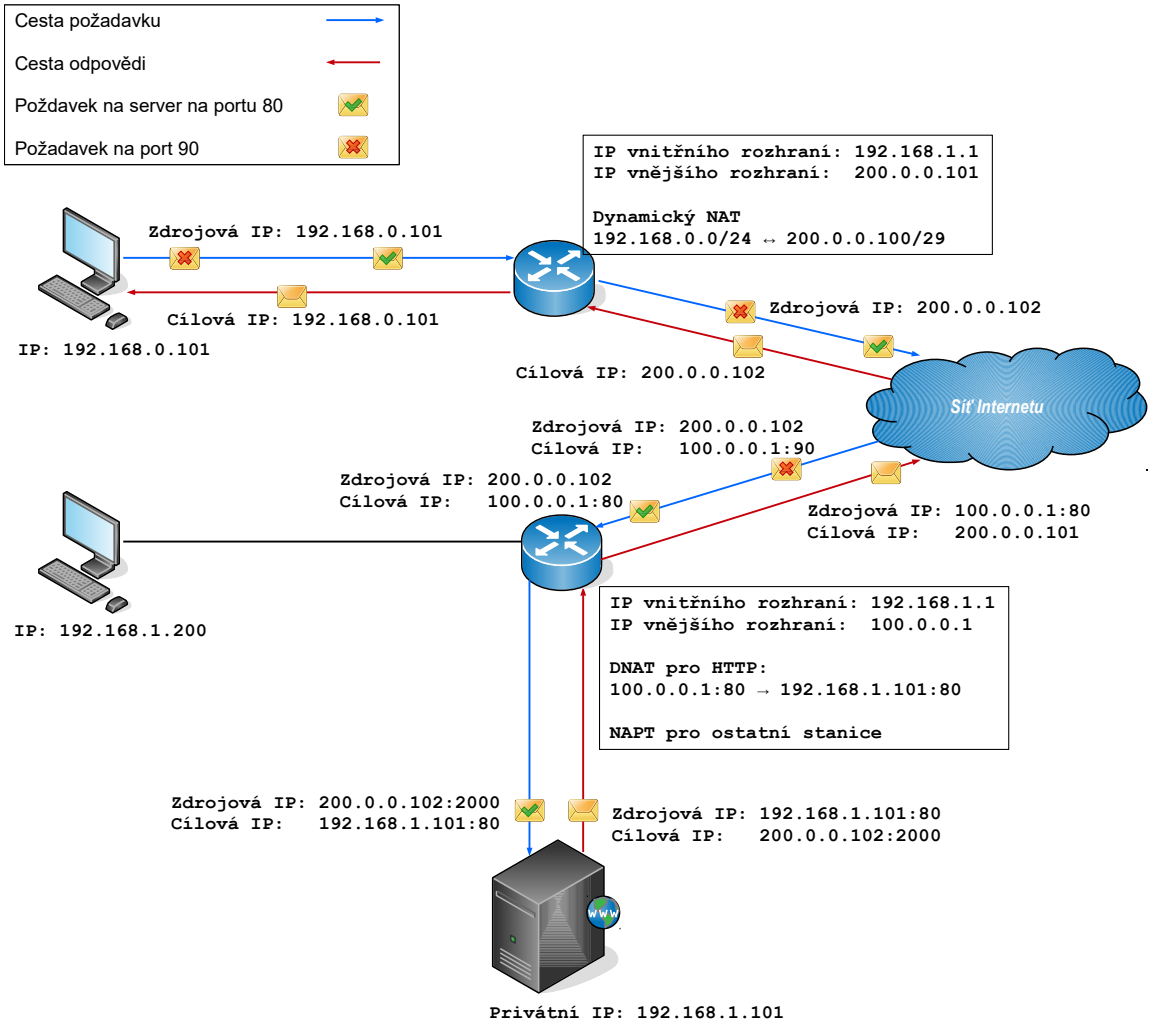
nemusí být stejné v čase. Nelze jej tedy použít jako jednoznačný identifikátor pro komunikaci zvenčí. Při pokusu o kontakt s touto adresou nemusíme kontaktovat danou stanici, případně kontaktujeme pouze vnější rozhraní brány. Obrázek 2.2 ilustruje komunikaci s využitím NAPT.



Obrázek 2.2: Průchod paketu skrze NAPT

Výše uvedené způsoby ukazují způsob komunikace, kdy chceme zpřístupnit veřejnou síť stanicím v síti privátní. Co ale v okamžiku, kdy potřebujeme zpřístupnit určité zařízení v privátní síti do veřejné sítě? Jedním způsobem je využít mapování 1:1. Tím vyhradíme specifický překlad pro danou stanici. Zpřístupníme tím však celý rozsah portů dané stanice. Pokud bychom tedy chtěli použít tuto metodu, tak musíme dané překladové pravidlo doplnit o příslušná pravidla pro firewall. Dalším nedostatkem této metody je nutnost samostatné adresy pro všechny takto zpřístupněné stanice.

Pokud tedy potřebujeme otevřít pouze určitou službu a nemáme k dispozici adresu, pak lze použít mechanismus Port Forwarding. Ten využívá NAT, přesněji DNAT (destination NAT) pro přesměrování provozu určité služby na stanici ve vnitřní síti. Provoz cílený na veřejné rozhraní brány je upraven přeposlán na tuto stanici. Ta pak může provoz zpracovat a odpovědět běžným způsobem. Nepotřebujeme tedy zvláštní adresu pro tuto stanici a firewall stačí nakonfigurovat pro „otevřené porty". Příklad použití DNAT pro Port Forwarding je ilustrován na obrázku 2.3.



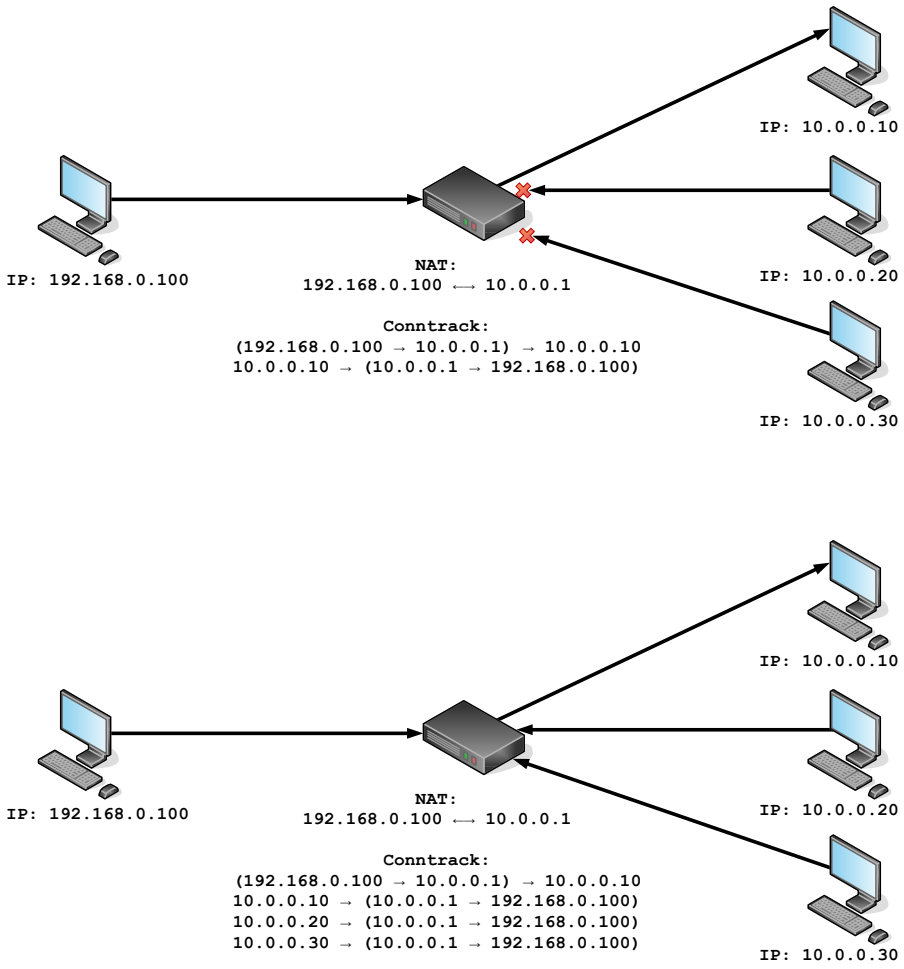
Obrázek 2.3: Příklad použití DNAT pro Port Forwarding

2.1.2 Mapování spojení a jejich chování

Během komunikace si NAT udržuje záznam o spojení a mapování mezi privátní a veřejnou adresou. Toto mapování zůstává živé během celé doby komunikace. V okamžiku uzavření toku je vazba odebrána a může být použita pro jinou komunikaci. Vlastní způsob tvorby těchto mapování se však může lišit v různých implementacích překladu. Častým způsobem identifikace je pětice identifikátorů zdrojová a cílová IP adresa, zdrojový a cílový port a protokol transportní vrstvy. Obecně rozlišujeme dva základní způsoby mapování zobrazené na 2.4:

Symetrické mapování V tomto mapování je každému toku přiřazen vlastní záznam. Zařízení vně sítě tedy mohou komunikovat pouze pomocí tohoto mapování. Jakákoli komunikace, která tomuto mapování nebude odpovídat, bude zahozena.

Full-Cone mapování V tomto mapování je možné komunikovat se zařízením v privátní síti pomocí mapované adresy od okamžiku jeho vytvoření. Dokud je mapování platné, pak je toto zařízení dostupné pro jakoukoli stanici ve vnější síti bez ohledu na to, zda bylo spojení otevřeno z privátní nebo vnější sítě.

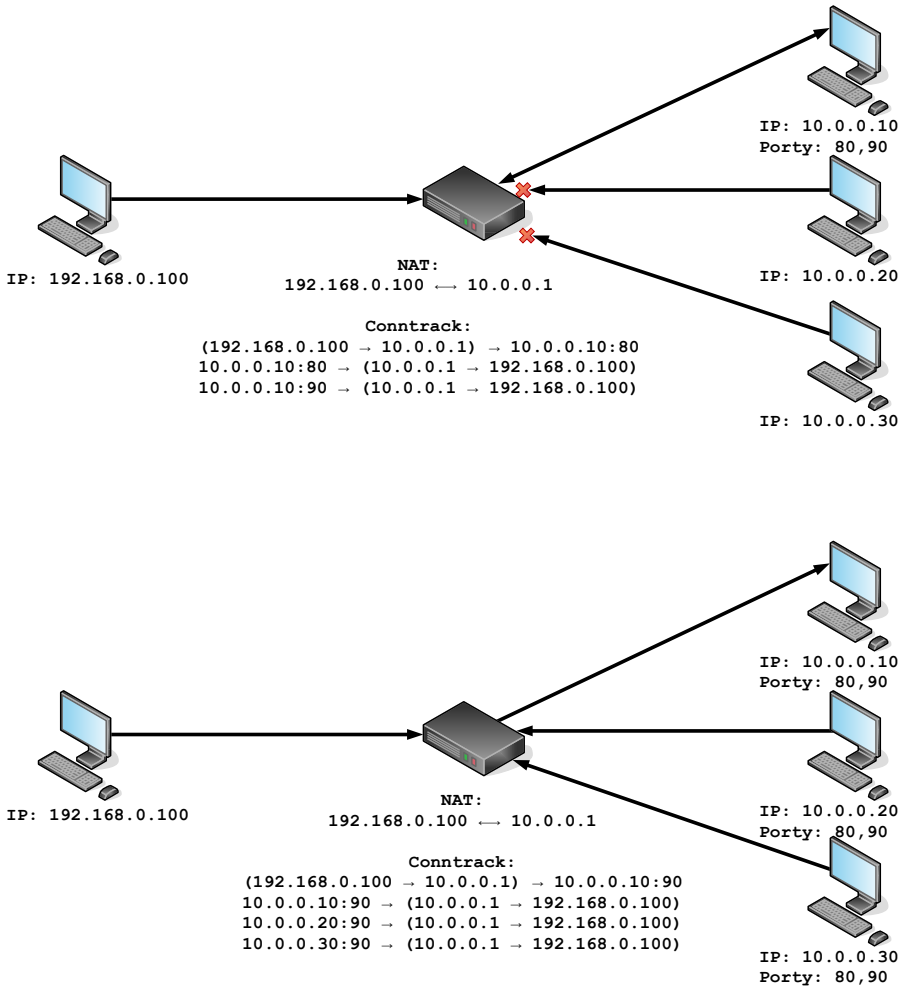


Obrázek 2.4: Způsoby mapování spojení: Symetrické a Full Cone

Způsob mapovaní se dále liší s ohledem na použitý transportní protokol. U TCP lze pozorovat chování odpovídající symetrickému překladu. To vychází z funkce samotného protokolu. V okamžiku kdy otevřeme TCP spojení z vnitřní sítě, pak je překlad vázán na toto spojení po dobu jeho životnosti. Tok dat tak může probíhat pouze mezi účastníky TCP. Jakákoli jiná komunikace na stejné identifikátory je zahazována. Protokol UDP naopak vykazuje spíše chování Full-Cone, jelikož UDP postrádá jakékoli informace o stavu spojení. S UDP nám navíc přibudou další možnosti pro komunikaci skrze NAT ilustrované na obrázku 2.5:

Restricted Cone mapování Při tomto mapování je komunikace otevřena pouze pro zařízení uvedené při vytvoření záznamu obdobně jako u symetrického překladu. Vnější strana však není omezena pouze na uvedený síťový tok, ale může otevřít spojení k zařízení v privátní síti na jiném portu.

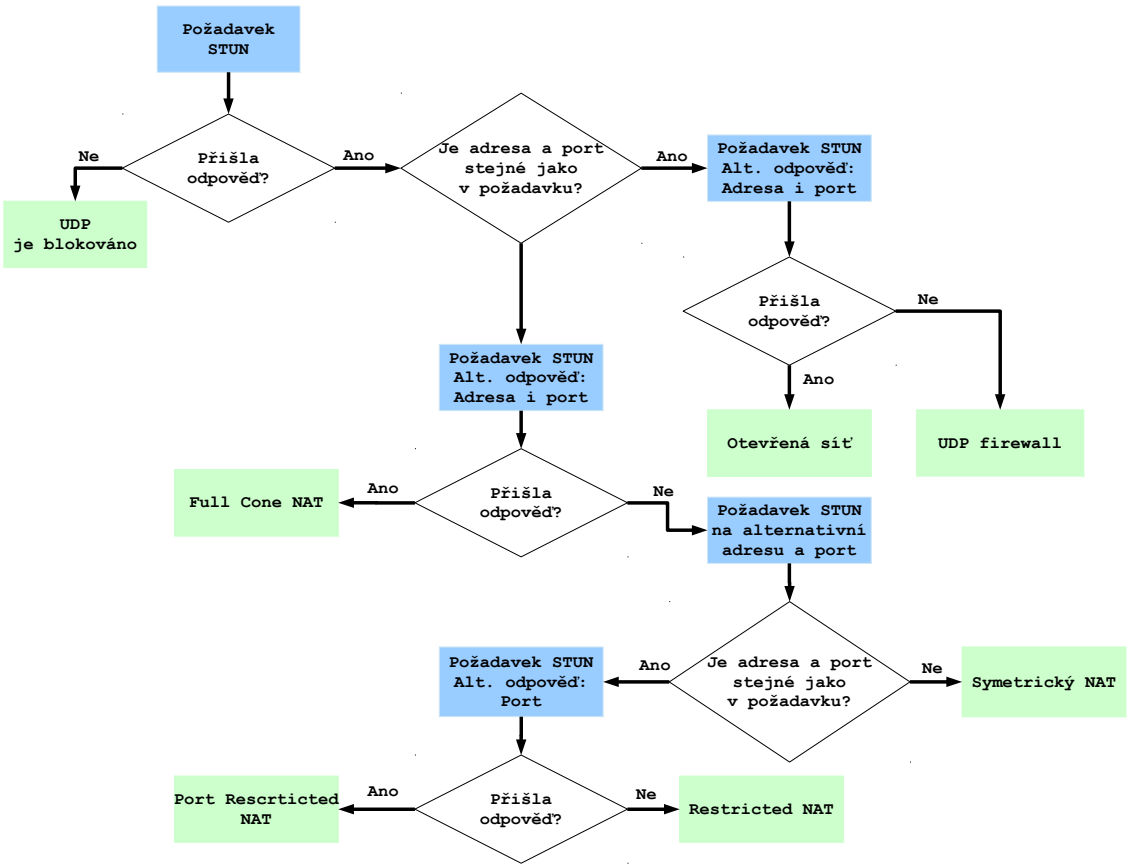
Port Restricted Cone mapování V tomto mapování je otevřena komunikace na určitém portu. Pokud se tedy připojí další stanice ve vnější síti na již otevřený port, pak tato komunikace proběhne. Spojení na jiné porty nebudou překládány.



Obrázek 2.5: Způsoby mapování spojení: Restricted a Port Restricted

Protokol UDP může být zpracováván všemi výše uvedenými způsoby s ohledem na implementaci NATu, který se nachází v cestě dat. Jak již bylo řečeno výše, implementace NATu se muže lišit v závislosti na zařízení nebo systému. Pro detekci způsobu zpracování je definován protokol STUN (RFC 3489 [16]). Ten je schopen detekovat pomocí dvou požadavků, zda je v datové cestě přítomen NAT a jaký je způsob zpracování spojení. V prvním požadavku odešle zprávu adresovanou přímo na zařízení. Pokud obdrží odpověď se stejnou adresou, pak se v cestě nenachází žádný NAT. Pokud je adresa jiná pak je odeslán druhý požadavek, ve kterém si klient vynutí odpověď pomocí adresy z prvního požadavku. Pokud se tato adresa shoduje, pak je v cestě Full-Cone NAT. V opačném případě je odeslána

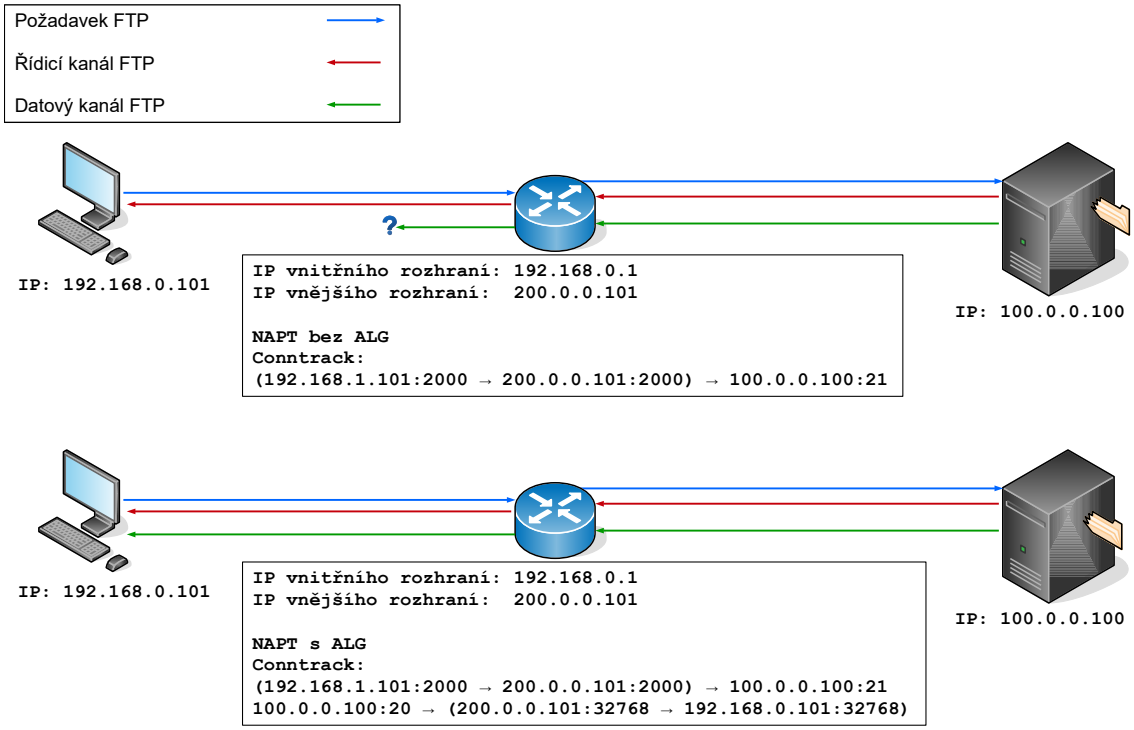
znovu první zpráva, ale tentokrát adresovaná na adresu z první odpovědi. Pokud obdrží stejnou odpověď, pak je NAT symetrický. V opačném odesílá znovu druhou zprávu a čeká na odpověď. Pokud odpověď dostane, tak komunikace probíhá přes Port Restricted Cone NAT, jinak se jedná o Restricted Cone NAT. Proces rozhodování je ilustrován na obrázku 2.6:



Obrázek 2.6: Rozhodovací proces mechanismu STUN

2.2 NAT a aplikační protokoly

Překlad adres je navržen k práci na síťové, případně transportní vrstvě. Tím je schopen spravovat běžnou komunikaci v síti. Existují však protokoly, které ke svému provozu potřebují další spojení. Tato spojení bývají otevírána z vnější sítě a jejich parametry jsou přenášeny v datech samotného protokolu. NAT však nemůže přeložit takto otevřené spojení, protože pro něj buď neexistuje pravidlo nebo není vytvořen záznam v tabulce spojení. Pokud by spojení prošlo nepřeložené, tak jej může cílová stanice zahodit. Pro tyto případy existuje pro NAT mechanismus ALG, neboli Application Level Gateway (RFC 3235 [17]). ALG zpracovává data aplikačních protokolů, u kterých probíhá k dohodě na dalším spojení a přidává dodatečné záznamy do tabulky spojení. Činnost ALG demonstruje obrázek 2.7 na protokolu FTP.



Obrázek 2.7: NAT s použitím ALG

V RFC 3027 [10] jsou definovány protokoly, které mohou způsobovat problémy při průchodu přes NAT. Podle něj lze protokoly rozdělit na takové, které nelze s překladem adres použít vůbec, a protokoly, které vyžadují další zpracování. Do první kategorie patří například protokoly IPSec (RFC 4301 [12], RFC 6071 [7]), Kerberos 4 a 5 nebo protokol RSH (Remote Shell). Do druhé kategorie patří převážně protokoly, které komunikují přes více spojení, jako je FTP v aktivním režimu nebo signalizační protokol SIP/SDP.

Dalším úkolem ALG je korektně udržovat stav hlaviček nižších vrstev, když dochází ke změnám do dat. To platí především pro textové aplikační protokoly. U nich je pravděpodobné, že změna dat způsobí změnu délky obsahu paketu. Tato změna musí být pak přenesena do příslušných položek v hlavičkách nižších vrstev. Ve všech případech je nutné přepočítat kontrolní součet paketu. V případě protokolu TCP je navíc nutné zohlednit tyto změny i do sekvenčních a potvrzovacích čísel. Ty jsou pro pakety vypočítány podle vzorce 2.1. Výjimkou jsou pakety na začátku a konci komunikace, kde se vždy přičítá 1 při přítomnosti příznaku SYN a FIN:

$$N_{ack+1} = N_{seq} + L_{pld} + F_{SYN} + F_{FIN} \tag{2.1}$$

N_{ack+1} ... Potvrzovací číslo následujícího paketu

N_{seq} ... Sekvenční číslo současného paketu

L_{pld} ... Délka obsahu dat

F_{SYN} ... Nastavený příznak SYN

F_{FIN} ... Nastavený příznak FIN

Protokol ICMP

Protokol ICMP (RFC 1812 [3]) vyžaduje po NATu další zpracování v případě, že obdrží chybový paket. Tyto pakety totiž ve svém těle přenášejí datový paket, který chybu vyvolal. NAT by měl tento zabalený paket použít pro nalezení spojení, ke kterému data patří, a podle něj provést patřičný překlad takto vloženého paketu. Pokud aktivní spojení neexistuje, je ICMP zahozeno. Stejný záznam pak NAT použije o pro překlad adres vlastního ICMP paketu. Podle RFC 5508 [19], jež definuje tento mechanismus zpracování, by NAT měl podporovat minimálně zprávy o nedosažitelnosti hosta, zprávy echo a reply a zprávy o přesáhnutí TTL.

Protokoly s odděleným datovým spojením

V protokolech, které mají oddělený datový kanál od řídicího, může dojít k problémům v situacích, kdy je datový kanál otevíraný ze strany serveru. Spojení nemusí být povoleno, pokud je NAT nastavený jako symetrický, nebo dochází k překladu portů. Tento jev se týká například protokolu FTP [14] nebo signalizačního protokolu SIP [15].

U protokolu FTP je možné se tomuto problému vyhnout použitím pasivního režimu. V tomto případě otevírá obě spojení klient. Vznikne pro ně tedy mapování pro překlad, se kterým může NAT pracovat běžným způsobem. Pokud bychom chtěli použít aktivní režim, pak musí NAT sledovat řídicí kanál FTP a upravit zprávu s příkazem PORT. Na základě této zprávy pak může vytvořit mapování pro datový kanál. Jelikož je FTP textově orientovaný protokol, tak s touto úpravou může dojít ke změně velikosti přenášených dat. NAT tedy musí korektně upravit i kontrolní součty. V případě, že by paket přesáhl velikost MTU, pak může úprava způsobit fragmentaci paketu. To může přinést problémy při další cestě paketu. Fragmentované pakety musejí totiž být znovu složeny, než budou zpracovány. Případně může dojít k zahození, pokud fragmentace není povolena.

Pro signalizační protokol SIP je nutné sledovat zprávy typu INVITE. V těchto zprávách je přenášena informace o dohodnutém komunikačním kanálu pro hlasová, případně obrazová data a adresy účastníků hovoru. Na základě obsahu těchto zpráv musí NAT vytvořit mapování pro přenos multimediálních dat protokolem RTP. Vzhledem k textové orientaci pro něj pak platí stejná pravidla ohledně fragmentace jako pro FTP.

Protokol IPSec

Protokol IPSec je v případě použití NAT nemožné nasadit, jelikož IPSec je navržený na kontrolu změn v paketových hlavičkách. V případě použití AH (Authentication Header) metody jsou informace pro kontrolu integrity vypočteny z původního paketu. Po průchodu NATem je však IP hlavička pozměněna. Kontrola na cílové stanici tedy selže a paket je zahozen. Pokud je použita metoda ESP (Encapsulating Security Payload), pak může dojít k problémům v okamžiku, kdy je nutné aktualizovat kontrolní součet v protokolu transportní vrstvy. Ten je však šifrovaný a NAT jej tedy nemůže změnit, aniž by poškodil data. Následkem toho je paket v cíli opět zahozen, tentokrát z důvodu neplatného kontrolního součtu. Je-li tedy požadováno nasazení protokolu IPSec, pak musíme jeho obsah zapouzdřit například pomocí GRE tunelu (RFC 1701 [8], 1702 [9] a 2784 [6]) nebo musíme IPSec nasadit pouze v části, kde není nutné provádět překlad adres.

Kapitola 3

Linux netfilter: implementace nástroje pro NAT systému Linux

Netfilter je framework pro filtraci a úpravu paketů používaný v systému Linux. Poskytuje sadu volání (netfilter), pro které je možné registrovat zpracovávající funkce, a nástroje pro specifikaci pravidel (iptables a nftables) a sledování spojení (conntrack-tools). Společně tvoří podstatnou část celého frameworku a zajišťují jak funkci firewallu, tak i překlad adres či jiné úpravy paketů. Pro účely práce se však budeme zabývat pouze moduly souvisejícími s překladem.

Celý program netfilter sestává z několika vrstev, přes které musí každý paket projít, než se dostane na výstupní rozhraní resp. bude zpracován samotnou aplikací. V každé vrstvě je paket „prosévánˮ přes tabulky určené k filtraci či úpravám. Pro účely práce nebudeme popisovat všechny tyto části, ale vybereme pouze ty stěžejní. Omezíme se tedy pouze na filtrovací tabulky a především tabulky určené k překladu adres. Budou nás tedy zajímat tyto části:

Tabulka RAW Tato tabulka je úplně první tabulkou v cestě paketu. Pravidla v ní obsažená ovlivňují sledování stavu spoejní pomocí celého nástroje netfilter. Lze například určitý provoz ze sledování stavu vynechat. Následkem tohoto pak tento provoz nebude také procházet tabulky NAT, což lze využít pro provoz, jehož stav nepotřebuje zařízení sledovat a nevyžaduje další úpravy.

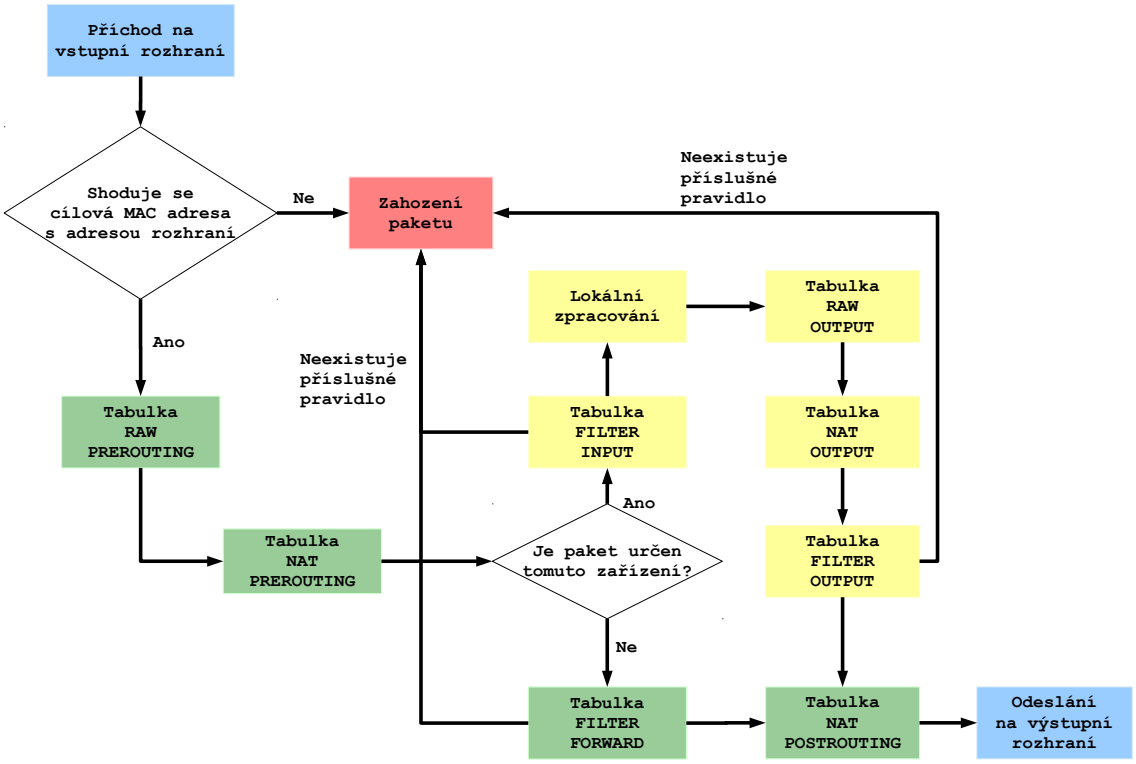
Tabulka FILTER Tato tabulka je určena k filtraci zpracovávaného provozu. Je používána jednak na vstupním rozhraní, ale také při vstupu do systému nebo výstupu ze zařízení. Pomocí této tabulky netfilter implementuje činnost firewallu.

Tabulka NAT Tato tabulka obsahuje pravidla pro úpravu paketů. Pomocí v ní uvedených pravidel provádí netfilter činnost překladu adres.

3.1 Průchod paketu přes netfilter

Jak je uvedeno v úvodu kapitoly, netfilter je složen z několika vrstev. V každé této vrstvě jsou používány výše specifikované tabulky a uložená pravidla. Všechny pakety zpracovávané zařízením musí projít touto strukturou ať už budou zpracovány lokálně nebo přeposlány dále. Struktura aplikace jako celku je komplikovaná a pro potřebu popisu průchodu není nutná v úplné podobě. Zjednodušíme ji tedy pouze na nejdůležitější tabulky. Průchod

paketu touto zjednodušenou strukturou je pak ilustrován na obrázku 3.1. Ten zachycuje tabulky, kterými musí paket projít a použité seznamy pravidel.



Obrázek 3.1: Průchod paketu skrze iptables

První tabulka, na kterou paket při vstupu do zařízení narazí, je tabulka RAW. Ta ovlivňuje pouze činnost modulu nf_conntrack, který je zodpovědný za správu záznamů o aktivních spojeních. Jediné záznamy, které bude tato tabulka obsahovat jsou tedy záznamy s akcí `NOTRACK`. Takový provoz nebude nijak zaznamenán v modulu nf_conntrack. Následkem toho tak bude sice stále podroben filtraci, ale již nebude možné takový provoz překládat. Překlad adres totiž vyžaduje, aby pro tok existovala položka v nf_conntrack. V opačném případě je po průchodu paket zpracován pomocí tabulky NAT. Ta provede překlad pro paket ještě před vyhledáním cíle ve směrovacích tabulce. Při nenalezení záznamu je paket ponechán ve stavu, v jakém byl obdržen.

Všechny pakety pak procházejí filtrací pomocí tabulky FILTER. Ta zastupuje funkci firewallu, a tedy zahodí veškerý provoz, pro který nemá záznam nebo je explicitně požadováno jeho zahození. Průchozí provoz je pak zpracován lokálně nebo je směrován dále. Pro tranzitní provoz je pak vyhledána cesta obvyklým způsobem pomocí LPM (Longest Prefix Match). Následně je tento provoz odeslán znovu do tabulky NAT a je vyhledáváno pravidlo pro překlad. Zpracovaný provoz pak opouští stanici nalezeným výstupním rozhraním.

Provoz generovaný samotnou stanicí prochází obdobnou cestou . Opět tedy prochází skrze tabulkami RAW, NAT a FILTER. Pro zpracování provozu však využívá jinou množinu pravidel, než pro vstupní a tranzitní provoz. O způsobu organizace pravidel budeme hovořit v následující sekci.

3.2 Specifikace a organizace pravidel

Pravidla pro v programu netfilter jsou organizována do seznamů nazývaných „chains". Každá tabulka obsahuje vždy sadu základních seznamů, které nelze odstranit. Pravidla do těchto tabulek lze pak vkládat přímo nebo je možné vytvořit vlastní seznamy pravidel a na ně se následně odkazovat. Vyhledání příslušného pravidla probíhá lineárně. Záleží tedy na pořadí, v jakém jsou vložena. Samotná pravidla jsou spravována pomocí nástroje iptables. My se soustředíme pouze na pravidla patřící tabulce NAT.

Tabulka NAT obsahuje čtyři základní seznamy, a to „PREROUTING", „INPUT", „OUTPUT" a „POSTROUTING". Jejich název odpovídá okamžiku, ve kterém jsou v nich uvedená pravidla vyhledávána a případně aplikována. Ze všech seznamů jsou obvykle upravovány pravidla v tabulkách PREROUTING a POSTROUTING. Jejich kombinací pak dosahujeme potřebného chování překladu. Případně je možné k překladovým pravidlům specifikovat pravidla pro filtraci. Následující příklady ukazují specifikaci pravidla pro překlad 1:1, 1:N a M:N, případně pro port forwarding:

```
# kombinace DNAT a SNAT pro stat. překlad 1:1
iptables -A PREROUTING -d 10.0.0.1 -j DNAT --to-destination 192.168.0.1
iptables -A POSTROUTING -s 192.168.0.1 -j SNAT --to-source 10.0.0.1

# překlad 1:N pomocí MASQUERADE
# při překladu se použije adresa rozhraní
iptables -A POSTROUTING -o eth1 -j MASQUERADE

# překlad M:N
iptables -A POSTROUTING -o eth1 -s 192.168.0.0/24 -j SNAT \
--to-source 10.0.0.0/24

#varianta pravidla pro konkrétní rozsah
iptables -A POSTROUTING -o eth1 -s 192.168.0.1-192.168.0.3 -j SNAT \
--to-source 10.0.0.1-10.0.0.3

# Port forwarding pro port 80 pomocí DNAT
iptables -A PREROUTING -d 10.0.0.1 -dport 80 -j DNAT \
--to-destination 192.168.0.1
```

3.2.1 Zpracování aplikačních protokolů

Pro zpracování aplikačních protokolů poskytuje sadu jaderných modulů. Požadujeme-li zpracování specifického protokolu, musíme zavést příslušný modul. Netfilter pak bude zpracovávat daný protokol podle pravidel nastavených v příslušných seznamech. Pravidla tedy není nutné nijak upravovat.

Kapitola 4

NAT implementovaný nad Intel DPDK

V době vzniku této práce je na FIT VUT v rámci projektu SProbe vyvíjen nástroj pro překlad adres implementovaný nad frameworkem Intel DPDK. Ten by měl být schopen dosáhnout vyššího výkonu při zpracování paketů, než konvenční řešení založená na Linuxu nebo jiných platformách. V kapitole bude nejdříve popsán vlastní framework a poté architektura samotného nástroje.

4.1 Framework Intel DPDK

Framework Intel DPDK [2] (Data Plane Development Kit) sestává ze sady knihoven. Ty poskytují rozhraní pro implementaci aplikací pro zpracování síťových dat. Mimo ně framework také dodává potřebné ovladače pro jádro systému. S těmito prostředky lze implementovat zpracování síťových úloh v userspace operačního systému. Je tak možné akcelerovat úlohy, jenž by zpracování v jádře systému zabraly více času. Tato zdržení mohou vzniknout jak kvůli nutné synchronizaci, tak i kvůli průchodu paketů přes moduly, které je nemusí zpracovávat. S frameworkem DPDK se tyto problémy sice přenášejí na programátora modulu, avšak je možné aplikaci pro zpracování zbavit nepotřebných částí nebo zjednodušit tok dat.

Samotný framework lze rozdělit do několika hlavních částí. Ty se starají jak o zprávu samotného hardwaru síťové karty, tak o správu paměti a plánování paketů. Tyto části si nyní stručně popíšeme.

Environment Abstraction Layer (EAL) Tento modul zajišťuje vytvoření pracovního prostředí pro DPDK aplikace. Jeho úkolem je přidělit požadované aplikaci potřebné systémové prostředky. Jedná se hlavně o síťová rozhraní, jádra procesoru a paměťový prostor. Vrstva EAL také poskytuje nástroje pro zpracování přerušení a synchronizaci.

Memory Pool Manager Modul pro správu paměti poskytuje rozhraní pro práci s pamětí v rámci aplikace. Pod jmenovaným memory poolem si můžeme představit statický prostor v paměti, jenž je vyhrazen pro datové struktury potřebné pro zpracování. Dále se snaží udržovat objekty v paměti zarovnané, aby aplikace mohla co nejlépe využít výkon pamětí.

Ring Manager Modul Ring Manager poskytuje rozhraní pro kruhové buffery. Tyto buffery mohou být využívány pro komunikaci v rámci aplikace samotné např. mezi vlákny nebo pro uchování dat pro další zpracování. Jejich implementace vychází z kruhových bufferů používaných v systému FreeBSD. Samotné buffery nevyužívají žádné zámky a umožňují i zpracování dat po dávkách.

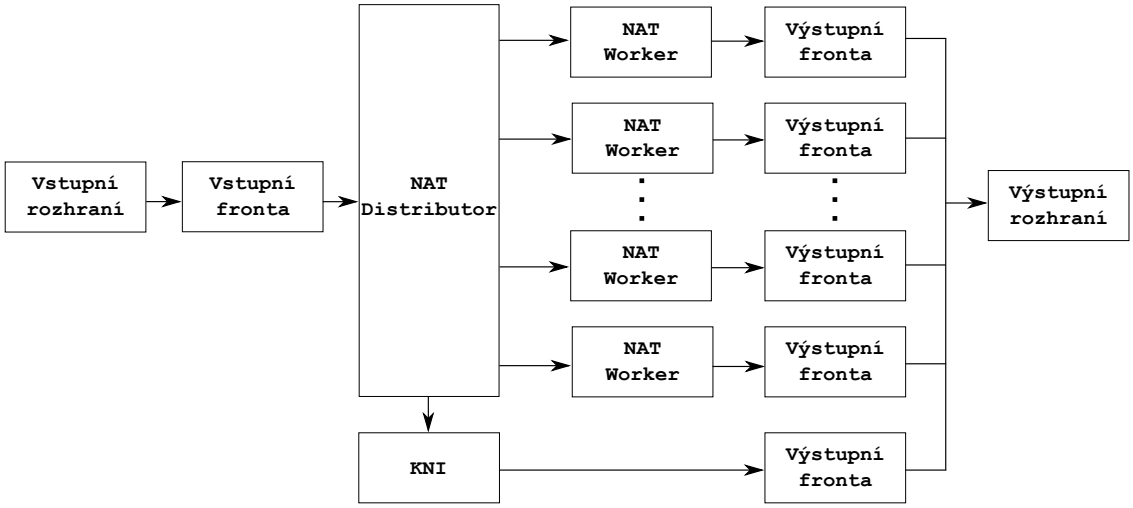
Network Packet Buffer Manager Poslední jmenovaný modul implementuje rozhraní pro práci se síťovými daty. Buffery vytvořené tímto modulem jsou primárně určeny pro zpracovávané pakety. Ty jsou v těchto bufferech ukládány v podobě struktury s metainformacemi o paketu, ke kterým je připojen vlastní paket. Mezi ukládané informace s patří například VLAN tag podle standardu 802.1q, výsledek hashe RSS (Receive-Side Scaling) nebo informaci o offloadu výpočtu kontrolních součtů v HW.

K výše jmenovaným pak ještě patří systémový ovladač rozhraní a knihovny pro podporu zpracování paketů. Systémový ovladač využívá API odvozené od userspace ovladače používaném ve FreeBSD. Pro co nejvyšší rychlost zpracování se namísto čekání na přerušení ovladač stále dotazuje síťového rozhraní na data. Jediná přerušení, která ovladač zpracuje jsou ta, která souvisí se změnou stavu linky. Ovladač je dostupný pro rychlosti linky 1, 10 a 40 Gb/s a podporuje nejen síťové karty společnosti Intel, ale také některé modely jiných výrobců (Melanox, Cisco, ...) a virtuální rozhraní. Díky své architektuře funkce rozhraní nesmí být volány paralelně. Knihovny pro zpracování paketů pak poskytují definice struktur pro zpracování síťových a transportních protokolů a algoritmy používané pro přepínání či směrování paketů.

4.2 Architektura DPDK NAT

Aplikace pro NAT nad DPDK je vyvíjena čistě jako nástroj pro překlad adres. To znamená, že aplikace velmi zjednodušuje některé funkce spolupracující s překladem. Mezi ně patří například základní směrování nebo filtrování paketů. Tyto funkce nejsou zapotřebí ve své plné podobě a výkon lze tedy primárně směrovat na překlad adres. Ostatní funkce jsou tedy ponechány na jádru systému a aplikace si pouze extrahuje potřebné informace.

Aplikace samotná pracuje v několika vláknech, kde jedno vlákno je vyhrazeno pro distribuci příchozího provozu. Další samostatné vlákno je pak vyhrazeno pro předávání dat do jádra systému. Toto vlákno zpracovává data, která neumí aplikace zpracovat sama, ale nelze je zahodit. Mezi tato data mohou patřit pakety cílené na zařízení, ARP pakety nebo směrovací informace. Zbylá vlákna jsou pak vyhrazena pro zpracování paketů. Celková architektura je ilustrována na obrázku 4.1. Kromě samotných částí aplikace je také ukázán tok dat.



Obrázek 4.1: Architektura DPDK NATu

4.2.1 NAT Distrubutor

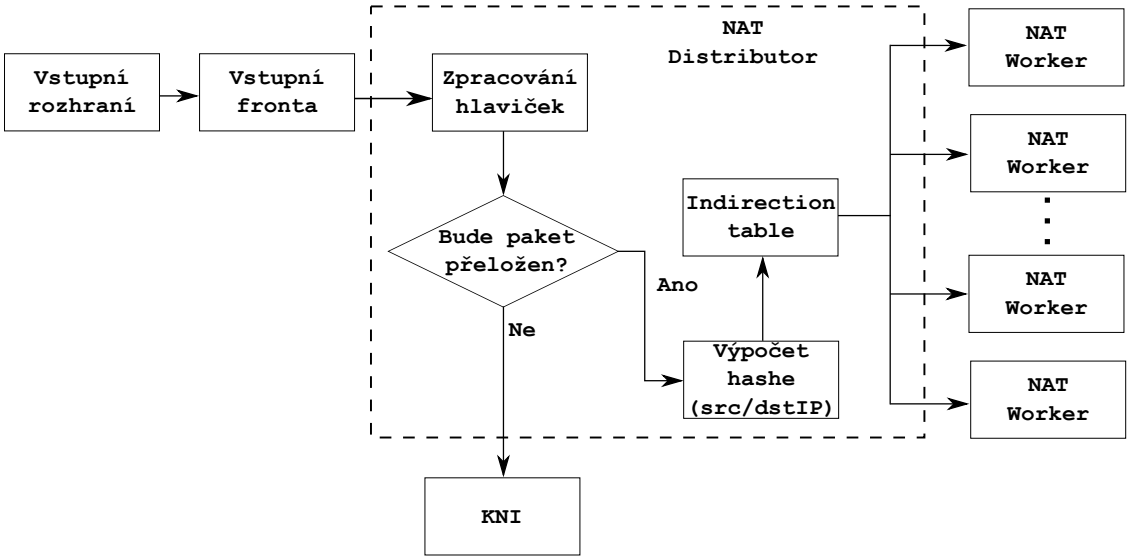
Abychom mohli zpracovávat provoz paralelně, je nutné provést jeho distribuci mezi jednotlivá vlákna. Tuto funkcionalitu zajišťuje běžně v jádře systém RSS ([20]). Ten pro rozložení zátěže mezi vlákna CPU vypočítá hash příchozího paketu a podle výsledku pak přidělí vstupní frontu paketu. Samotná hash je vypočítána ze čtveřice složené z IP adres a portů transportního protokolu. Přiřazení front datovým tokům je pak uloženo v tabulce „indirection table". Pro výběr fronty se používá spodních sedm bitů vypočítané hashe.

Pro účely překladu však potřebujeme, aby obě strany provozu byly zpracovány stejným vláknem. Tabulky s pravidly jsou totiž rozděleny mezi vlákna NAT worker. Zpracování paketů mezi různými vlákny by tak nebylo možné, jelikož by se překládala pouze jedna polovina celého datového toku. Systém RSS sice umožňuje změnu způsobu výpočtu hashe, ale vždy bere nejméně obě IP adresy. To by produkovalo nežádoucí chování. NAT distributor tedy implementuje vlastní obdobu RSS, kde rozděluje provoz pouze na základě jedné IP adresy, a to zdorjové v případě odchozího provozu z privátní sítě a cílové adresy v opačném směru. Tímto lze zaručit, že datový tok bude v obou směrech zpracován stejným vláknem NAT worker. NAT distibutor tedy musí implementovat vlastni indirection table a mechanismus distribuce zpracovávaných paketů. Dalším úkolem NAT distributora je také zpracování dat, jež nebudou procházet překladem. Tato data buď zahazuje, nebo je předává do rozhraní KNI[1]pro zpracování jádrem systému. Architektura NAT distributora je ilustrována na obrázku 4.2. Průchod paketu pak probíhá následovně:

1. Paket je přijat na vstupním síťovém rozhraní.
2. NAT distributor provede extrakci informací z hlaviček. Pokud se jedná o provoz, který nebude překládán (směrovací informace, IPv6 pakety aj.), data jsou poslána rozhraním KNI do jádra systému.
3. Ze získané hlavičky proběhne výpočet hashe na základe zdrojové IP, pokud se jedná o odchozí provoz z privátní sítě, nebo cílové IP adresy, pokud se jedná o příchozí provoz.

[1]Kernel Network Interface

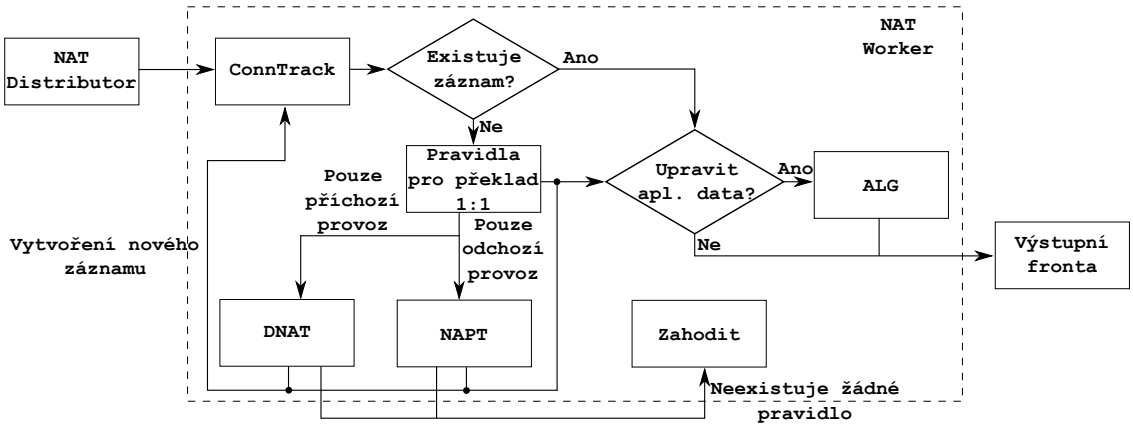
4. Na základě výpočtu hashe je podle záznamu v indirection table vybráno vlákno NAT worker, jež zpracuje paket.



Obrázek 4.2: Distribuční modul DPDK NATu

4.2.2 NAT Worker

Modul NAT worker zajišťuje veškerou činnost spojenou s překladem adresy. Jsou v něm uložena pravidla pro překlad, právě aktivní spojení i přístup k ALG. Pro jeho běh může být vyhrazeno tolik vláken, kolik je dostupných jader CPU po vyhrazení prostředků pro distribuční modul a rozhraní jádra systému KNI. Na osmijádrovém CPU tedy může běžet až šest těchto vláken. Každé vlákno si pak udržuje vlastní informace o pravidlech a aktivních spojeních. Zpracovávaný provoz je do těchto NAT workerů směrován právě z modulu NAT distributor. Architekturu ilustruje obrázek 4.3.



Obrázek 4.3: Worker modul DPDK NATu[2]

Pro každý paket, který není zpracován pomocí KNI, je nejprve vyhledán záznam v tabulce aktivních spojení. Pokud takový záznam existuje, je paket přeložen podle něj a je odeslán na výstup, případně do ALG. V opačném případě je nutné takový záznam vytvořit na základě pravidel dostupných pravidel. Nejdříve NAT worker prohledá pravidla pro překlad 1:1. V případě, že nenalezne, pak v závislosti na rozhraní, kde byl provoz zachycen, prohledá tabulku pro NAPT, nebo pro DNAT. První použije pro provoz odcházející z privátní sítě, druhou v opačném směru. Pokud není pravidlo nalezeno v žádné tabulce a neexistuje záznam, je paket zahozen. Jinak je vytvořen záznam pro nové aktivní spojení a paket je po překladu odeslán.

Všechna vlákna si udržují svoji tabulku aktivních spojení. Ta obsahuje všechny toky, které dosud prošly skrze NAT a byly přeloženy. V současnosti je tato tabulka využívána především jako „cache" pro již probíhající spojení, aby nebylo nutné hledat správné pravidlo pro každý zachycený paket. V porovnání s modulem nf_conntrack je tedy značně zjednodušena. Záznam pak přetrvává v tabulce do doby, než vyprší jeho neaktivní timeout nebo není spojení aktivně ukončeno např. příznakem FIN u protokolu TCP.

4.2.3 Konfigurace a specifikace pravidel

Konfigurace DPDK NATu je v současné fázi podporována pouze ve statické podobě. Pravidla tedy musí být specifikována dopředu a při spuštění budou načtena. Tato pravidla již nelze nijak změnit. Současně s nimi je načítána také konfigurace parametrů celého nástroje. Mezi tyto parametry patří např. velikost bufferů pro zpracovávané pakety nebo kapacita tabulky spojení. Díky tomu, že se jedná o userspace aplikaci, je možné konfiguraci snáze organizovat. Pro porovnání, netfilter sice poskytuje nástroj pro konfiguraci za běhu, ale parametry jednotlivých modulů jako je conntrack jsou více rozptýleny v souborovém systému. Část těchto parametrů se nachází v `/proc` systému, části související s parametry přijímáním a odesíláním paketů jsou uloženy v adresáři `/sys`.

Statická konfigurace je pro DPDK NAT rozdělena do dvou souborů, konfiguračního souboru a souboru s pravidly (ukázka specifikace je uvedena v příloze E). V konfiguračním souboru jsou uložena nastavení velikosti tabulky spojení, vyhrazení zdrojů pro jednotlivá rozhraní a velikosti tabulek pro uchování pravidel. Soubor s pravidly pak specifikuje jednotlivá pravidla, která rozděluje do čtyř skupin:

1. Pravidla pro překlad NAPT
2. Pravidla pro unrestricted NAT 1:1
3. Pravidla pro restricted NAT 1:1
4. Pravidla pro DNAT

První skupina specifikuje pravidla pro dynamický překlad způsobem NAPT. Jejím obsahem tedy definujeme seznam adres, na které mohou být překládána odchozí spojení. Uvedené adresy musí být jiné, než je adresa výstupního rozhraní.

Skupina 2 a 3 specifikuje pravidla, která budou využívána k překladu 1:1. Ty jsou záměrně rozděleny do dvou podskupin, a to restricted a unrestricted. Pravidla ve skupině unrestricted umožňují průchod dat z obou stran NATu. Je tedy možné se pomocí adresy

[2]Příchozí a odchozí provoz je označen vzhledem k privátní síti. Odchozí tedy znamená, že paket opouští privátní sít, příchozí naopak vstupuje do této sítě

uvedené pro překlad na takovou stanici připojit z vnější sítě. Restricted pravidla tuto funkcionalitu neumožňují. To znamená, že provoz splňující takové pravidlo, bude propuštěn pouze v případě, že existuje odpovídající spojení z privátní sítě. V opačném případě bude provoz zahozen.

Poslední skupina pak specifikuje pravidla pro DNAT, resp. port forwarding. Pomocí v ní uvedených pravidel můžeme zpřístupnit pouze určité porty, směřující na uvedenou adresu.

4.2.4 Přídavné moduly

Základní implementace podporuje pouze běžně používané metody překladu. Není tedy schopná zpracovávat aplikační protokoly nebo provádět pokročilejší zpracování. Pro tyto účely je implementováno rozhraní pro přídavné moduly. Ty jsou do hlavní aplikace nahrávány formou pluginů případně podobně jako moduly pro netfilter. Lze je tedy buď specifikovat při startu nebo nahrát za běhu aplikace. V současnosti je pomocí těchto modulů implementována funkce ALG pro protokoly FTP a SIP.

Kapitola 5

Návrh testování síťových zařízení

V předchozí kapitole jsme se věnovali vlastní funkci nástroje pro překlad síťových adres, způsobům nastavení a jeho aplikaci v praktické komunikaci. V této kapitole se zaměříme na testování zařízení, která by měla být určena k provozování těchto nástrojů. V kapitole bude nejdříve nastíněna problematika testování síťových zařízení. Stručně popíšeme, jaké parametry budeme sledovat při testech výkonu či funkčnosti. Na jejich podkladě pak navrhneme testovací metodiky, které lze aplikovat na testy nástrojů pro NAT. Ty budou zkoumat jak výkonnost samotného překladu, tak funkční vlastnosti nástroje. Budeme tedy sledovat, zda jsou provedené překlady správné nebo jestli nedochází k výpadkům spojení při činnosti NATu.

5.1 Obecné metody testování a měření síťových zařízení

Při návrhu a vývoji jakéhokoli nástroje pro zpracování síťových dat je cílem vytvořit takovou implementaci, jež bude provádět svůj úkol správně a zároveň co nejefektivněji. V případě síťových zařízení se tedy zaměřujeme převážně na veličiny jako je propustnost, tedy kolik dat jsme schopni zpracovat, nebo odezva komunikace. Ta nám pomáhá zjistit, jaký vliv bude mít tento nástroj na běžnou komunikaci, jestli například nezanese do přenosu dat nežádoucí zpoždění aj. Vzhledem k tomu, že síťová zařízení často nemají možnost výměny hardwarových komponent, je vhodné se zaměřit také na nároky CPU a paměti.

5.1.1 Měření výkonnosti

Pro měření propustnosti jsou často používány postupy popsané v RFC 2544 [5]. Ten specifikuje, jaké délky paketů mají být při měření použity, či jakým způsobem má být generovaný provoz. Dále také popisuje podmínky, v jakých má být testování prováděno, jak má být nakládáno s případným nežádoucím provozem, jež se může v testovacím zapojení objevit, např. ARP pakety, LLDP (Link Local Discovery Protocol [1]) pakety apod. V RFC 6815 [4] jsou pak upřesněny podmínky testování a určení metod popsaných ve výše zmíněném dokumentu. Při měření je možné a také vhodné testovat zařízení i v extrémních situacích, abychom zjistili, jaké jsou maximální možnosti těchto zařízení. Musíme však dbát na to, že může dojít k neočekávaným stavům.

5.1.2 Funkční testování

Při funkčním testování bychom se měli zaměřit hlavně na to, zda zařízení vykonává svou činnost správně v různých situacích, ve kterých se může ocitnout. Zde se mohou sledované vlastnosti lišit podle účelu daného nástroje nebo zařízení. Testujeme-li například směrovač, pak je pro nás důležité, zda opravdu vybírá správné cesty, zda vyhledává pomocí nejdelšího prefixu nebo jestli správně nakládá s více záznamy pro stejná data. V případě firewallu bychom zase sledovali, jestli opravdu filtruje pouze provoz, který má uvedený v tabulkách. Dále můžeme opět sledovat chování v neočekávaných situacích, např. při přetížení zařízení velkým množstvím vstupních dat nebo větším počtem souběžných úloh.

Při funkčním testování můžeme na testované zařízení nahlížet buď způsobem black-box nebo white-box. V prvním případě nás nezajímá, jak vypadá vlastní tok dat, ale zda dojde k očekávanému výsledku. V případě směrovače očekáváme doručení na požadovanou stanici, při překladu adres navázání spojení a jeho bezproblémový průběh. Při white-box testování nás zajímají i další parametry. Mezi ně může patřit přesná činnost zařízení nebo struktura provozu během zpracování. Například velké množství poškozených nebo opakovaně přenesených dat na jednom rozhraní může například ukazovat na chybu při zpracování paketů, která se nemusí při běžném přenosu dat nijak projevit.

5.2 Aplikace testovacích metod na nástroje pro překlad adres

V předchozí sekci jsme nastínili obecné způsoby, jak provádět měření síťových zařízení a nástrojů, a také jejich funkční testování. Nyní tyto postupy aplikujeme na nástroj pro překlad adres. Zájmovými výkonnostními parametry bude propustnost těchto nástrojů a doba cesty paketu (Round Trip Time) při průchodu tímto zařízením. Z hardwarových statistik budeme sledovat uvedené zatížení CPU a využití paměti. U CPU můžeme měřit statistiky i na úrovni přerušení, pokud nám to dovolí nástroje přítomné v OS na zařízení, nebo budou získávaná data dostatečně vypovídající.

5.2.1 Metody pro výkonnostní testování

Pro měření výkonnosti zpracování využijeme výše popsanou topologii zapojení v jejím plném rozsahu. Při měření budeme sledovat hlavně počet zpracovaných paketů a dále dobu přenosu paketu (Round Trip Time, dále jen RTT) mezi klientem a serverem. Data pro měření propustnosti nám může poskytovat buď samotné NAT zařízení nebo generátor paketů, pokud je schopen měřit množství příchozího provozu. RTT bude měřit klient v simulované síti.

V první metodě testu se zaměříme na výkon překladu adres. Do tabulky vložíme pravidlo, které bude pokrývat generovaný provoz. Ten bude mít nastaveny takové parametry, aby co nejrovnoměrněji zatížil všechna jádra procesoru, která zpracovávají příchozí provoz. Poté se upraví směrovací tabulky, aby byla funkční požadovaná komunikace a provoz nemohl projít mimo testovací prostředí. Pokud by generovaný provoz opustil testovací zapojení, mohl by mít vliv na případné výsledky nebo by mohl způsobit problémy v živé síti.Dále se upraví pravidla v ARP tabulkách, aby nedocházelo k zahazování provozu nebo k nežádoucímu broadcastu. Komunikace je znázorněna na obrázku 5.2. Vlastní test pak bude probíhat následovně:

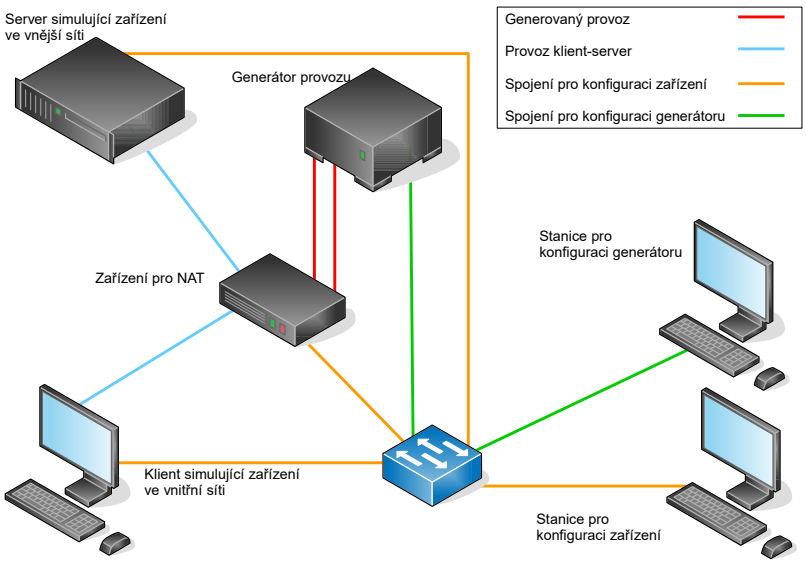
1. Generátor začne vytvářet provoz pro zatížení NATu na zadaném vytížení linky.

2. Po uplynutí předem stanoveného času je odebrán vzorek výstupu generátoru a výstupu NATu. Tyto hodnoty udávají, kolik paketu je ve skutečnosti vysláno do vnější sítě.

3. Klient postupně vyšle pomocí traceroute paket serveru a podle obdržené odpovědi změří RTT. Tuto činnost opakuje vícekrát, pokaždé s jinou IP adresou. Změna adresy na straně serveru je nutná k tomu, abychom předešli jakémukoli cachování na straně NATu.

4. Po změření RTT je generátor zastaven, NAT vymaže veškeré záznamy o známých připojeních, generátoru je zvýšená intenzita generování paketu a procedura se opakuje od bodu 1. Test končí, když je změřeno poslední RTT a generátor je nastaven na maximální možné zatížení linky.

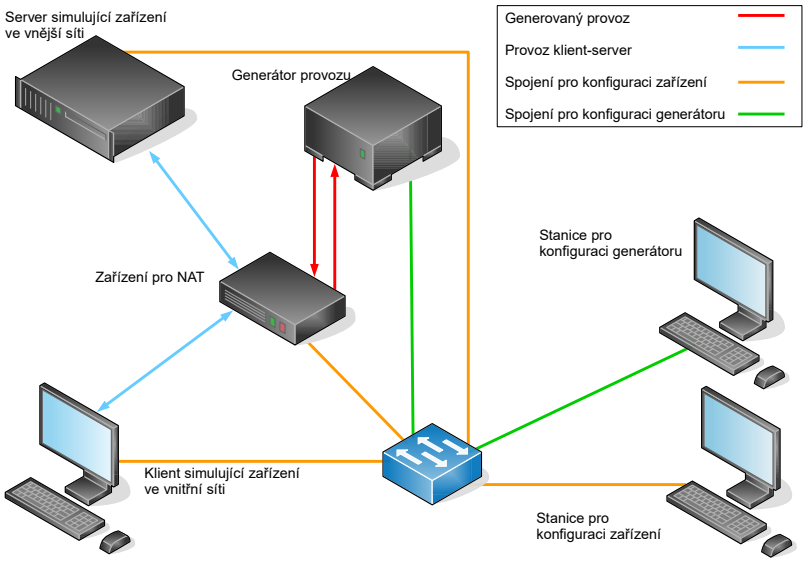
5.2.2 Popis zapojení pro testování a měření



Obrázek 5.1: Topologie pro testování zařízení NAT

Na obrázku 5.1 je ilustrováno zapojení, které je v současné době připraveno na testování a měření. Topologie je navržena tak, aby bylo možné testy a měření co nejlépe automatizovat a zároveň bylo možné co nejblíže simulovat reálné zapojení zařízení, kde klient je v síti s privátním adresováním a server je běžně dostupný ze sítě internetu. Během testu potřebujeme NAT vytížit dalším provozem pro zpracování. Ten nám pomáhá generovat specializované zařízení (v současné době se jedná o Spirent TestCenter SPT-2000A). Celá topologie je zapojena tak, aby žádný testovací provoz nemohl být puštěn do běžné provozní sítě. Pokud by tedy zařízení musela být spojena přes přepínač, je nutné vyhradit vlastní VLAN. Propojení s generátorem provozu by za každých okolností mělo být přímé. Případný přepínač by nemusel zvládnout plné vytížení spojení navíc v současné topologii není k dispozici přepínač pro rychlosti vyšší než 1 Gbit.

Zabarvení jednotlivých spojnic v topologii ukazuje, jaký provoz po těchto spojích bude procházet. Červené spoje obsahují pouze data z generátoru. Ta jsou po průchodu NATem vrácena zpět, jelikož generátor je schopen i analýzy dat. V měření je však převážně využíván pro zjištění počtu přijímaných paketů. Modré spoje pak ukazují cestu simulovaných uživatelských dat mezi klientem a serverem. Ostatní spojení jsou určena pouze pro monitorování a konfiguraci jednotlivých prvků. Stanice určená pro správu generátoru je v současnosti vyhrazená zvlášť z důvodu dostupného softwarového prostředí. Pokud to však bude možné, pak se o konfiguraci generátoru i všech ostatních zařízení bude starat jedna stanice.

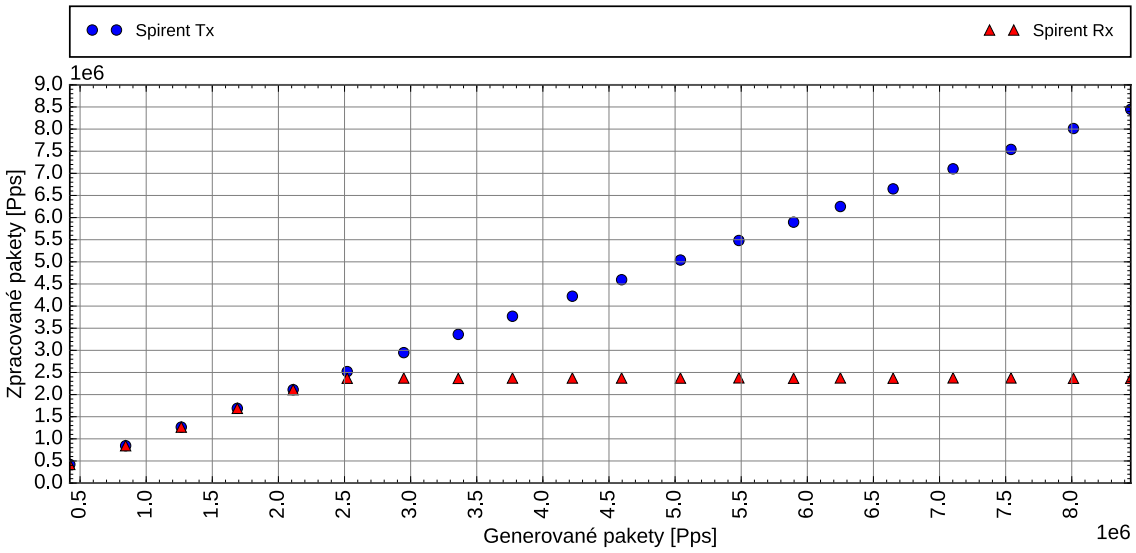


Obrázek 5.2: Topologie pro výkonnostní testování

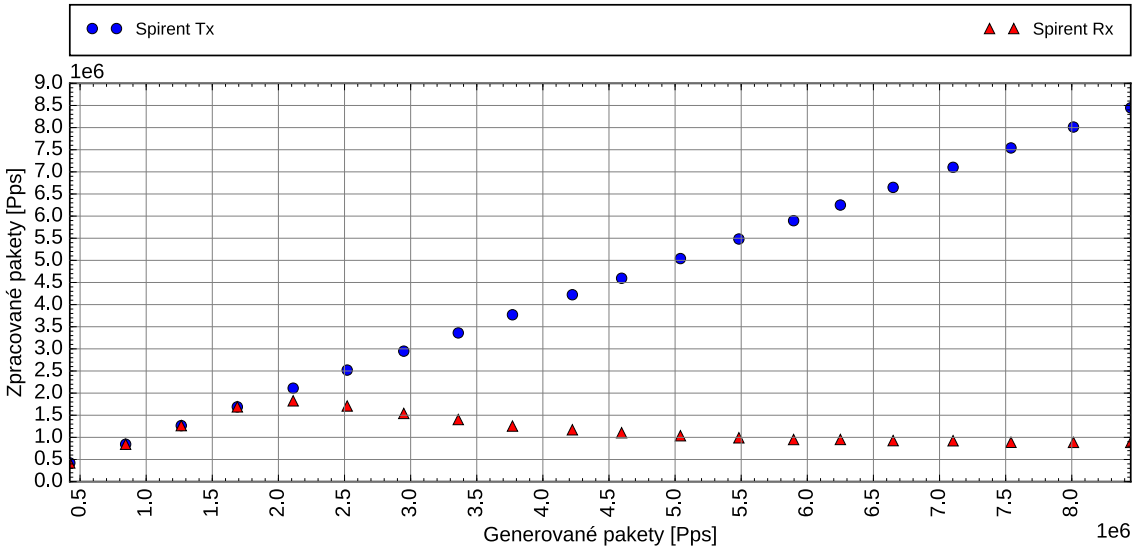
Pokud nám to zařízení umožní, můžeme kromě propustnosti a RTT měřit také paměťové a procesorové nároky. Mezi zajímavé statistiky procesoru patří především úroveň softwarových přerušení (Soft Interrupts). Tato přerušení vznikají, když dochází k přijímání nebo odesílání paketů. Můžeme tedy sledovat, kolik procent času tráví procesor právě zpracováváním těchto přerušení a kolik jinou činností. Měření těchto hodnot se však zatím ukazuje jako obtížné, jelikož nástroje dostupné pro Linux, jako je např. mpstat (součást balíčku sysstat[1]), hlásí tyto hodnoty velmi nepřesně. Tato nepřesnost se projevuje hlavně v okamžicích, kdy dostupná síťová karta má více front pro zpracování paketů a každou frontu zpracovává jiné jádro procesoru.

Výkonnost mohou ovlivnit i nastavení podpůrných modulů pro NAT (conntrack) nebo způsob zpracování přerušení. Je například docela významný rozdíl, pokud máme v conntrack volný prostor pro vytvoření nového záznamu pro spojení nebo již musíme některé položky z tabulky vyřadit. Vliv tohoto nastavení je vidět na grafech 5.3 a 5.4. Modrou barvou je vyjádřeno množství generovaného provozu a červenou pak skutečný počet přijatých paketů. První graf ukazuje situaci, kdy má conntrack dostatek místa pro uložení nových toků. V tomto případě zařízení při dosažení maximálního výkonu tuto hodnotu udržuje. Pokud však conntrack zaplníme, bude počet zpracovávaných paketů klesat s vyšším zatížením linky, jako ukazuje druhý graf.

[1] http://sebastien.godard.pagesperso-orange.fr/



Obrázek 5.3: Graf propustnosti skrze NAT s volným místem v conntrack



Obrázek 5.4: Graf propustnosti skrze NAT s přeplněnou tabulkou conntrack

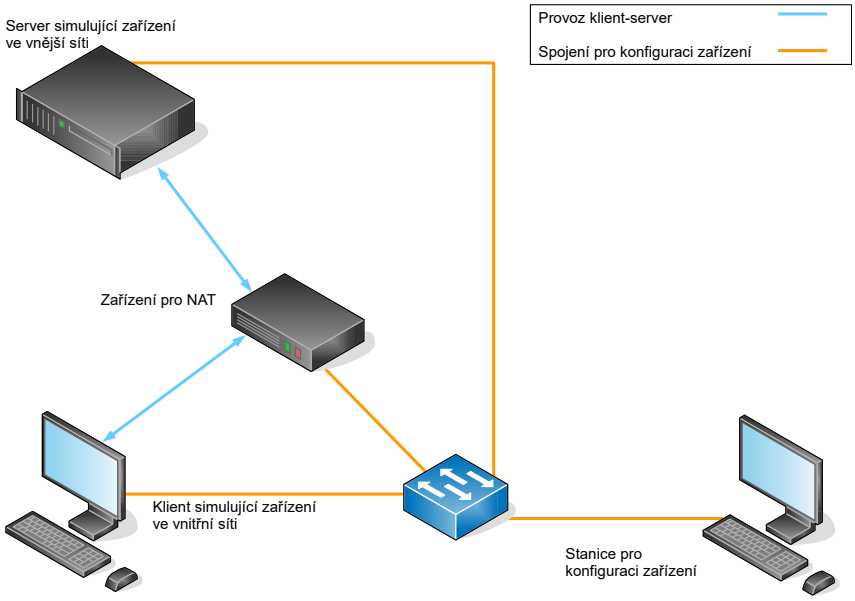
Dalším parametrem ovlivňujícím výkon je způsob zpracování přerušení. Pokud nám síťová karta umožní využívat více front pro příjem, resp. odesílání dat, pak záleží, zda jsou všechna přerušení zpracována na jednom jádře resp. procesoru, nebo na více. Vliv může mít i samotné hardwarové zapojení. Zde může docházet v případě víceprocesorových systémů ke komunikaci mezi procesory. To muže negativně ovlivnit výkonnost zpracování. Při testování je nutné všechny tyto vlivy zohlednit a provést testy v různých konfiguracích, aby bylo možné odhalit nejvýhodnější nastavení pro maximální výkonnost.

5.2.3 Funkční testování překladu

Při funkčním testování je pro nás důležité, zda překlad probíhá očekávaným způsobem. Jeho výsledek se projeví především tím, že jsme schopni při správné konfiguraci překladu navázat spojení. Navázané spojení pak nesmí být během komunikace přerušeno a musí protékat správná data. Zpracování různého typu provozu můžeme nasimulovat pomocí právě replikací uživatelských činností (stažení/nahrání souboru, volání pomocí VoIP aj.). Všechny testy pak můžeme provádět jak metodou black-box, tak white-box.

Při black-box testování se zaměříme pouze na úspěšné provedení simulované činnosti bez ohledu na průběh komunikace. Pokud jsme tedy schopní komunikovat skrze NAT podle nastavené konfigurace, pak považujeme úkon za úspěšný.

Při white-box testování se pak zaměříme i na samotný průběh přenosu. Bude nás tedy zajímat zachycený obsah komunikace a pokud nám to implementace NATu dovolí, tak i obsah tabulek spojení. V zachycené komunikaci pak budeme sledovat obsah paketů, především položky, kde očekáváme změnu. Dále můžeme sledovat počet opakovaných přenosů či paketů odeslaných v chybném pořadí. Jejich výskyt by měl být co nejmenší. Výjimkou je situace, kdy emulujeme vyšší ztrátovost nebo záměrně necháme měnit pořadí paketů. Pak by se chyby měly vyskytovat přibližně v rozsahu nastavených parametrů.



Obrázek 5.5: Topologie pro verifikaci funkce NAT

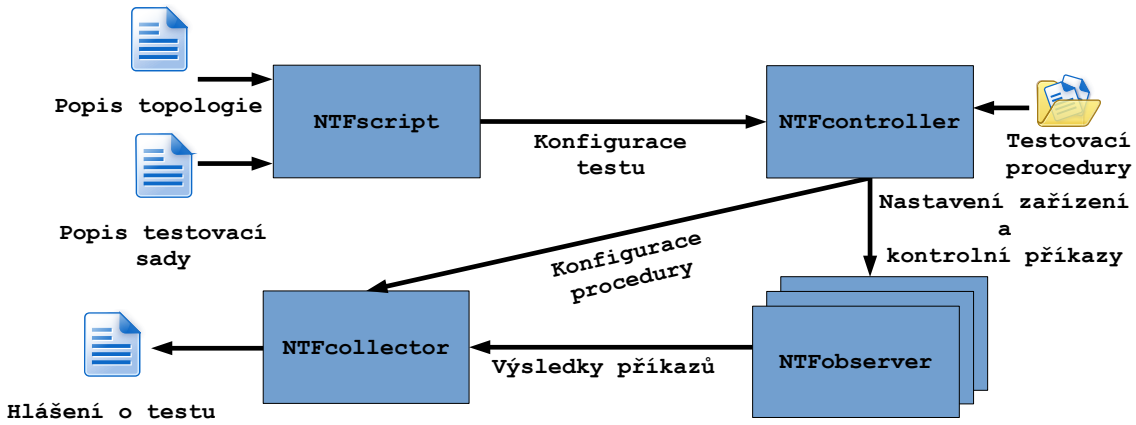
Kapitola 6

NTF - framework pro testování zařízení NAT

V předchozí kapitole byly specifikovány metody pro testování, které budeme aplikovat na prověřovaná zařízení. Následující kapitola popisuje nástroj NTF (NAT Testing Framework) implementovaný v rámci diplomové práce. Uvedený nástroj je navržený pro automatizaci testování nástrojů pro NAT. V kapitole bude popsána celková architektura nástroje, implementace jednotlivých komponent a způsob užití.

6.1 Architektura nástroje

Framework je navržen jako sada spolupracujících modulů. Ty zajišťují zpracování vstupních skriptů, konfiguraci zařízení a vykonání definovaných příkazů a sběr dat. Podle činnosti jsou odvozeny i názvy modulů: NTFscript, NTFcontroller, NTFcollector a NTFobserver. První tři jmenované slouží právě k interakci s uživatelem, konfiguraci zařízení a agregaci výsledků. Přestože mohou všechny moduly běžet odděleně, budou tyto tři modul často spuštěny na jedné stanici. NTFobserver je pak přítomen mna testovacích stanicích a vykonává příkazy, které obdrží od řídicí stanice. Finální architektura frameworku NTF včetně komunikačních kanálů je ilustrována na obrázku 6.1.

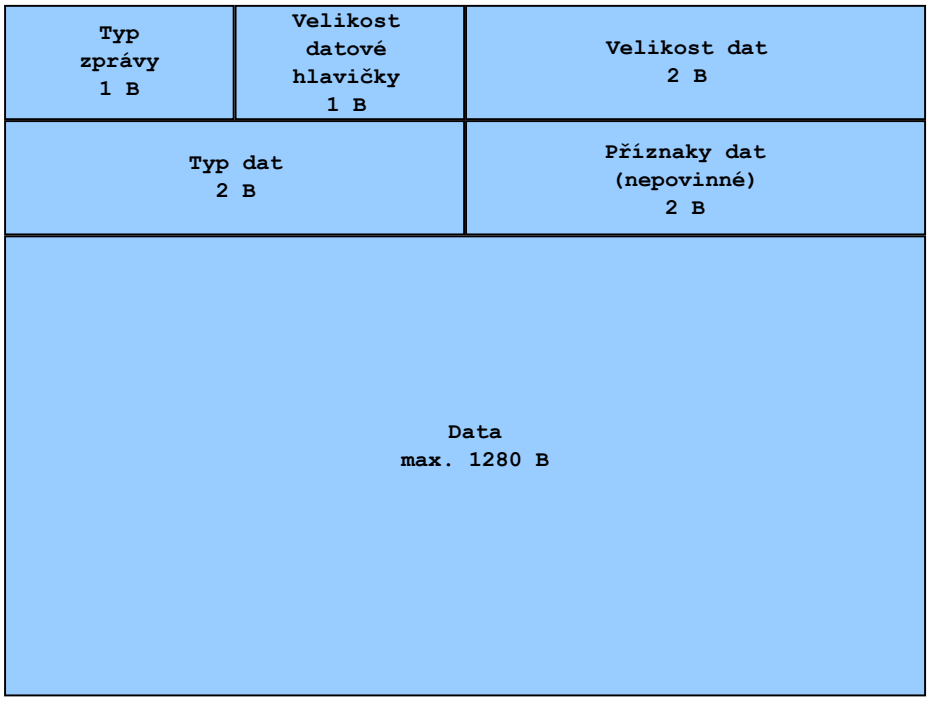


Obrázek 6.1: Architektura testovacího nástroje

6.1.1 Komunikace mezi moduly a zařízeními

Jak již byl zmíněno v úvodu, framework sestává z nezávisle spouštěných modulů. To umožňuje distribuovat moduly mezi více zařízení. Příkladem může být oddělení NTFcollectoru na vlastní stanici pro uchování dat. Stanice pro NTFcontroller pak může být pouze jediná v testovacím prostředí a uživatelé potřebují pouze vstupní rozhraní NTFscript. Moduly NTFobserver však musí být povinně přítomny na všech testovaných zařízeních, u kterých požadujeme automatickou konfiguraci a vykonávání příkazů v rámci testů. Modul samozřejmě musí být možné na dané zařízení nainstalovat.

Pro účely komunikace těchto modulů pak tak byla implementována knihovna poskytující komunikaci pomocí socketů a to buď BSD s TCP protokolem nebo UNIX. Zároveň s ní byl vytvořen jednoduchý komunikační protokol. Pomocí něj jsou v rámci frameworku předávány zprávy o jednotlivých příkazech, konfigurace zařízení či výsledky testů. Implementace tohoto protokolu je součástí souborů `protocol.cpp` a `protocol.hpp`. Tyto soubory definují celkový formát zprávy ilustrovaný na obrázku 6.2 a poskytují funkce pro zpracování a generování těchto zpráv. Formáty dat jsou pak definovány v souborech `msg_<typ_zprávy>.hpp`.



Obrázek 6.2: Formát zprávy `Protocol::msg_t`

Přestože komunikační rozhraní využívá TCP, tak všechny moduly samy řeší potvrzování přenesených zpráv. Každá přenesená zpráva je tedy patřičně potvrzována pomocí ACK zprávy definované v protokolu. Při spuštění a připojení se navíc moduly musí identifikovat pomocí 4-way handshake metody. Ta je buď provedena za pomoci HANDSHAKE zpráv, pokud potřebujeme pouze jednoduché připojení anebo pomocí HELLO zpráv. Ty přenášejí navíc konfigurační adresu zařízení. Na HELLO zprávu je pak odpovědí HANDSHAKE zpráva nebo druhá HELLO zpráva. Všechny podoby 4-way handshake jsou uvedeny v příloze A.

6.1.2 Průběh testu

1. Spustíme skriptovací rozhraní s popisem topologie a testovací sady.
2. Skriptovací rozhraní odešle konfiguraci do kontroléru.
3. Kontrolér převede obsah skriptu do objektové podoby.
4. Kontrolér čeká na připojení monitorovacích modulů
5. Testovaná zařízení se připojí ke kontroléru.
6. Kontrolér po ukončení čekání ověří, zda jsou potřebná zařízení připojena.
7. Kontrolér rozešle konfigurační parametry testovaných zařízením podle načtené topologie.
8. Kontrolér a monitorovací pokusí připojit ke kolektoru. Pokud neuspějí, monitorovací moduly nebudou odesílat výsledky. V opačném případě se monitorovací moduly identifikují kolektoru a kontrolér odešle konfigurační zprávu s procedurou.
9. Kolektor vytvoří hlavičku hlášení o testu a čeká na připojení všech monitorovacích modulů.
10. Kontrolér zahájí proceduru uvedenou načtenou v testovací sadě. Během procedury postupně odesílá příkazy specifikovaným zařízením. Se zahájením odešle kolektoru zprávu, aby nečekal na další monitorovací moduly. Pokud některý monitorovací modul není připojen, pak jeho výsledky nebudou zahrnuty.
11. Po ukončení procedury odešle kontrolér zprávu o konci testu do kolektoru a všem monitorovacím modulům.
12. Monitorovací moduly dokončí rozpracované příkazy, provedou základní vyhodnocení a odešlou výsledky do kolektoru. Po odeslání propagují zprávu o konci testu získanou z kontroléru.
13. Kolektor agreguje a vyhodnotí obdržené výsledky a dokončí hlášení o testu.

6.2 NTFscript: vstupní skriptovací rozhraní

Vstupní rozhraní frameworku zajišťuje definici topologie testovacího zapojení a specifikaci testovací sady. Jejich popis je rozdělen do dvou samostatných souborů. Tím je možné použít stejnou topologii pro různé sady testů a naopak. Pro popis vstupních hodnot využívá modul skriptovací jazyk Lua[1]. Rozhraní samotné pak poskytuje sadu funkci implementovaných v jazyce C++, které zajišťují přenos konfigurace do řídicího modulu.

Skript definující topologii obsahuje popis zapojení zařízení, které jsou potřebné pro provedení testů. Definuje zařízení a jejich konfigurační rozhraní, síťová rozhraní použitá během testů a zapojení na úrovni linkové vrstvy. Specifikace jednotlivých stanic obsahuje i jméno, které je využíváno ve skriptech definujících testovací proceduru. Tyto skripty budou popsány jako součást kontrolního modulu. Lze také nastavit, zda je na stanici přítomen monitorovací modul (NTFobserver), a také, zda má být stanice automaticky konfigurována.

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

Potřebná zařízení lze tedy nastavit ručně a modul NTFobsrver lze použít pouze pro spouštění potřebných příkazů případně lze zařízení z monitorování vyjmout úplně.

Při popisu síťových rozhraní je společně s IP adresou definována i MAC adresa, která bude použita pro specifikaci statických ARP záznamů, pokud budou nutné (plánované rozšíření). Dále je možné nastavit, zda chceme zachytávat provoz procházející daným rozhraním, nebo zda je rozhraní považováno jako primární. To je použito v případě, že v topologii není definován spoj stanic na úrovni linkové vrstvy. V opačném případě je použito přímé spojení. Jako poslední lze nastavit příznak, že rozhraní je připojeno do WAN. Toto označení lze využít na NAT stanici pro specifikaci rozhraní ve vnitřní a vnější síti. Tato označení pak budou použita při autokonfiguraci NATu (plánované rozšíření).

Skript pro definici testovací sady pak obsahuje konfiguraci cest k testovacím procedurám. Uvedené procedury budou spuštěny nad topologií specifikovanou v předchozím skriptu. Provedení procedur probíhá v režii řídicího modulu. Skript také může obsahovat parametry upravující některé příkazy prováděné v proceduře. Mohou to být např. timeout pro spojení nebo počet opakovaných pokusů pro znovunavázání v případě neúspěchu. Skript by pak měl být zakončen příkazem `start_test()`, který spustí testovací sadu.

6.3 NTFcontroller: modul pro řízení testu

Modul NTFcontroller zajišťuje zpracování konfigurace testu, nastavení příslušných zařízení a provedení testovacích procedur. Modul sestává ze dvou vláken, kde první přijímá popis testovacího scénáře ze vstupního rozhraní ve formě zpráv. Na jejich základě sestaví reprezentaci topologie, se kterou budou testy pracovat. Dále zpracuje testovací procedury a jejich parametry. Po zpracování probudí druhé vlákno, které začne provádět proceduru. V průběhu testu pak modul zamítá nová připojení do doby, než je test dokončen.

Plánovací vlákno modulu zajišťuje zpracování vstupu získaného ze skriptovacího rozhraní. Během své činnosti vlákno přijímá zprávy, ze kterých sestaví objektovou reprezentaci topologie. Postupně tedy vytváří potřebné stanice a přiřazuje definovaná rozhraní. Po sestavení topologie vytvoří na základě druhého vstupního skriptu seznam prováděných testů včetně potřebných parametrů. Po obdržení příkazu pro start testu, vlákno uzavře konfiguraci a probudí exekuční vlákno. Po dobu běhu exekučního vlákna pak odmítá další vstupní data, aby nedošlo k poškození konfigurace. Nový test je tedy možné začít až po dokončení právě probíhající sady.

Exekuční vlákno provádí konfiguraci stanic v testovací topologii a vykonání vlastní testovací procedury. Po probuzení od plánovacího vlákna vyčkává na připojení od zařízení, která mají mít aktivní moduly NTFobserver. Pokud tedy nejsou všechna zařízení vyřazena z konfigurace, pak čeká alespoň na jedno z nich. V případě ostatních čeká pouze po určitou dobu. Po ukončení připojování resp. vypršení čekací doby vlákno ověří, zda jsou zařízení přítomna. Pokud některé z nich chybí, jsou aktivní stanice odpojeny a test končí neúspěchem.

Po ověření přítomnosti aktivních zařízení vlákno provede konfiguraci potřebných rozhraní. Nastaví tedy na příslušná rozhraní správné adresy. Následně přidá výchozí cestu vedoucí přes NAT resp. směrovač sloužící jako brána do vnitřní sítě. Cestu přidává na základě existence spojení s tímto zařízením. Pokud by tedy bylo více menších sítí, tak budou výchozí brány přiřazeny na základě spojení definovaného v topologii. Po dokončení konfigurace vlákno začne provádět samotnou testovací proceduru.

Testovací procedura je stejně jako popis topologie a sady testů definována v jazyce Lua. Procedura popisuje příkazy, které budou vykonány jednotlivými zařízeními. Příkazy lze

spouštět synchronně (jeden po druhém) nebo lze spustit sadu příkazů souběžně. V případě takto spuštěné dávky si pak lze pomocí příkazu `sync_all_devices()` vynutit synchronizaci topologie. Modul odešle synchronizační zprávu a čeká na odpověď od všech sledovaných stanic. Tu obdrží v okamžiku, kdy jsou všechny spuštěné příkazy ukončeny. V definici procedury lze také spustit případně zastavit záchyt dat. Ten bude spuštěn na všech zařízeních, kterým byl v topologii nastaven příznak záchytu. Po ukončení provádění skriptu vlákno odešle ukončovací zprávu. Tou si vynutí synchronizaci topologie a zároveň upozorní kolektor, že má očekávat zbylé výsledky. Po korektním ukončení může NTFcontroller přijímat další testovací vstupy.

6.4 NTFobserver: modul pro konfiguraci a ovládání zařízení

Úkolem modulu NTFobserver je konfigurovat příslušné zařízení zúčastněné v testu a provádět příkazy na základě testovací procedury. Modul by měl být spuštěn na všech stanicích, které jsou použity v testu a nejsou explicitně definovány jako nesledované nebo nejsou vyřazeny z autokonfigurace. Modul se také stará o záchyt dat na příslušných rozhraních a sběr dat o provedených příkazech.

Při spuštění se modul nejdříve pokusí připojit k běžícímu modulu NTFcontroller. Po úspěšném navázání spojení se modul identifikuje adresou konfiguračního rozhraní, což by mělo být to, které bylo definováno v topologii. Po identifikaci se modul pokusí o stejný úkon s modulem NTFcollector. Pokud toto spojení nebude úspěšné, pak modul neodešle data z testu. Průběh jednotlivých příkazů však bude vždy zaznamenávat na dané stanici. Po ukončení připojení čeká modul na příkazy. Ty mohou buď provádět konfiguraci nebo vyvolávat akce pro přenos dat, či vytváření spojení. V případě konfigurace rozhraní si modul zaznamená, které rozhraní bylo nastavováno, aby bylo možné po ukončení testu vymazat jeho konfiguraci a bylo vytvořeno čisté prostředí pro další test.

Pro provádění veškerých úkonů modul využívá nástrojů dostupných v operačním systému. Jejich sadu je NTFobserver schopen nastavit při spuštění. Modul pak přebírá po provedení úkonu návratový kód pro příkazy, pro které je tato možnost dostupná. S jejich pomocí pak vytvoří záznam o provedení. Jeho součástí je cílová adresa, cesta k souboru, pokud je dostupná, všechny příznaky nastavené pro daný příkaz a dále získaný výsledek a očekávaný výsledek.

Jakmile je test dokončen, modul provede vymazání použitých konfigurací a vyčká na dokončení zbylých probíhajících příkazů. Poté projde záznamy o vykonaných příkazech a na základě návratových hodnot vyhodnotí výsledek provedení. Ten je zapsán do souboru na stanici. Záznam může sloužit pro základní vyhodnocení, avšak nebere ohled na očekávanou hodnotu výsledku. Ta je vyhodnocena až v modulu NTFcollector. Během tvorby tohoto souboru odesílá záznamy právě do modulu NTFcollector, kde budou výsledky agregovány a zhodnoceny podrobněji (plánované rozšíření). Po této činnosti modul odešle zprávu o dokončení testu do kolektoru a ukončí se, nebo zůstane v běhu a očekává další příkazy.

6.5 NTFcollector: modul pro agregaci dat

Poslední implementovaný modul má za úkol agregovat výsledky z modulů NTFobserver a provést jejich vyhodnocení. V současnosti modul provádí pouze binární vyhodnocení provedených úkonů. V plánovaném rozšíření by pak měl být schopen i analýzy zachyceného provozu a zohlednění dalších parametrů v hodnocení. Podrobněji budou tyto parametry ro-

zebrány v kapitole 9. Modul je opět složen ze dvou funkčních bloků, konfiguračního a agregačního. Jejich činnost podobně jako v kontroléru vykonávají oddělená vlákna.

Konfigurační vlákno komunikuje s kontrolérem a očekává údaje o prováděném testu. Na jejich základě vytvoří začátek hlášení o testu. Po jeho vytvoření čeká na dokončení konfigurace a spuštění testu. Během této činnosti se mohou připojovat testovací stanice pomocí NTFobserver modulu. Ty spravuje agregační vlákno. Jakmile obdrží vlákno startovací zprávu, konfigurační vlákno přeruší připojovací smyčku agregačního vlákna a do ukončení testu bude zamítat další zprávy, podobně jako vstupní rozhraní frameworku.

Agregační vlákno během konfigurace stanic čeká na připojení od NTFobserver modulů. Připojení je podobně jako v kontroléru identifikováno konfigurační adresou stanice. Po identifikaci je komunikaci vyhrazeno vlákno, aby bylo možné sbírat data souběžně. Jakmile je test spuštěn, vlákno přestane přijímat nová spojení a vyčká na dokončení testu. Komunikační vlákna mezitím sbírají zprávy s výsledky od stanic do doby, než přijde ukončovací zpráva. Po přijetí této zprávy od všech stanic začne vlákno zpracovávat získané výsledky. Pro každý příkaz je vypočtena hodnota úspěšnosti na základě porovnání očekávaného výsledku s dosaženým. Pro stejné úkony jsou tyto výsledky agregovány a výsledná známka sečtena. Po agregaci výsledků vlákno vypočítá celkovou známku součtem všech dílčích známek. Dosažené výsledky jsou pak zapsány do souboru se záznamem. Ten obsahuje mimo již vytvořené hlavičky a celkové známky také detailní výpis prováděných příkazů. Záznam testu tedy může vypadat například takto:

```
TEST REPORT ON RES_UNRES
DATE: 24-04-2016 15:15
SUMMARY MARK: 3/6
========================================================================
192.168.0.249 -- ICMP echo to 172.16.1.5 (Failure)                 [ 1/1 ]
192.168.0.249 -- L4 protcol echo to 172.16.1.5 using UDP (Failure) [ 1/1 ]
192.168.0.249 -- L4 protcol echo to 172.16.1.5 using TCP (Failure) [ 1/1 ]
192.168.0.254 -- ICMP echo to 172.16.1.100                         [ 0/1 ]
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using UDP         [ 0/1 ]
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using TCP         [ 0/1 ]
========================================================================
```

V hlavičce souboru je uvedeno jméno procedury, jež odpovídá názvu skriptu s touto procedurou. Následuje datum a čas provedení a výsledná známka. Zbytek záznamu je výpis provedených příkazů podle zařízení. Ta jsou identifikována svou konfigurační adresou. V záznamu příkazu je pak uvedeno, o jaký úkon se jedná, kdo byl jeho cílem a parametry. Poslední je dílčí hodnocení úspěšnosti.

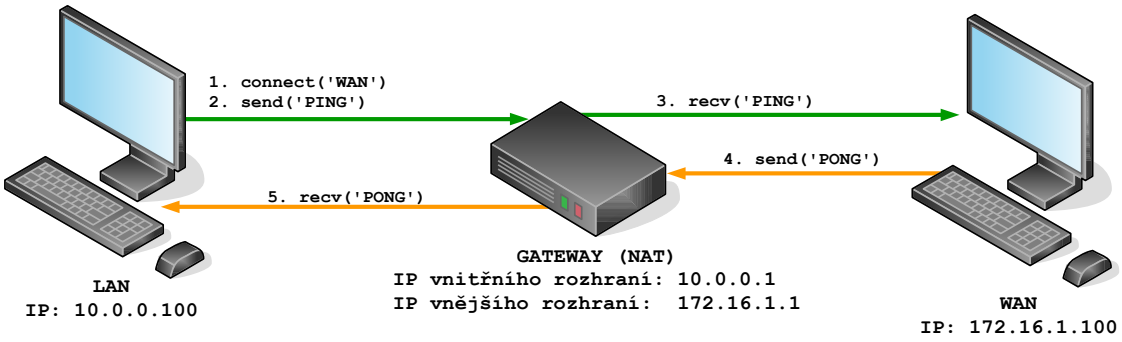
Kapitola 7

Testovací scénáře a jejich aplikace

V kapitole 5 byly popsány metodiky výkonnostního a funkčního testování včetně jejich aplikace na NAT. Kapitola 6 pak popisuje nástroj, který k tomuto testování bude využit. V následující kapitole si tedy popíšeme scénáře, ve kterých budeme NAT testovat. V popisu si projdeme, jakou činnost budeme sledovat a jak ji budeme vyhodnocovat.

7.1 Scénář 1: Navázání spojení skrze NAT

V prvním testovacím scénáři se pokusíme o navázání spojení skrze NAT. K tomu použijeme mechanismus vestavěný do monitorovacích modulů. Ty při spuštění naslouchají na nastaveném portu. Během testu zařízení v privátní síti provede pokus o připojení k serveru a vyšle textový řetězec "PING". V případě úspěšného doručení zprávy by měl server odpovědět řetězcem "PONG". Spojení je vyhodnoceno jako neúspěšné, pokud se nelze připojit nebo zařízení v pozici klienta neobdrží odpověď do deseti sekund. Obrázek 7.1 ilustruje jednotlivé kroky scénáře.

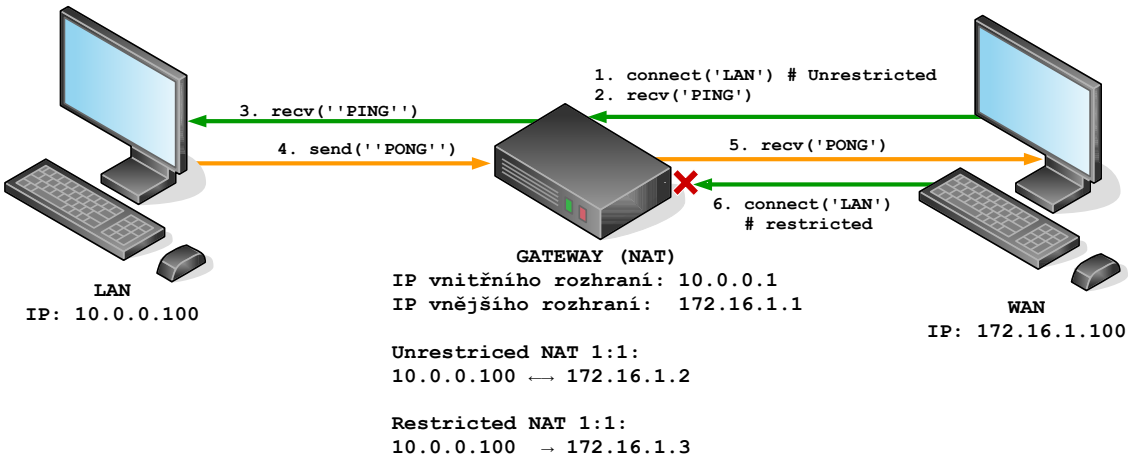


Obrázek 7.1: Testovací scénář 1: Navázání spojení skrze NAT

7.2 Scénář 2: Restricted vs. Unrestricted pravidla

Zařízení NAT v mnohých aplikacích spolupracuje s firewallem jako zabezpečovacím prvkem. Právě s pomocí firewallu můžeme komunikaci ovlivnit tak, aby přijímala pouze spojení, která jsou otevřena z privátní sítě. Pro NAT tedy nastavíme pravidla pro překlad 1:1. Pro jedno z nich přidáme zmíněné pravidlo pro firewall. V testu pak postupně zkusíme navázat

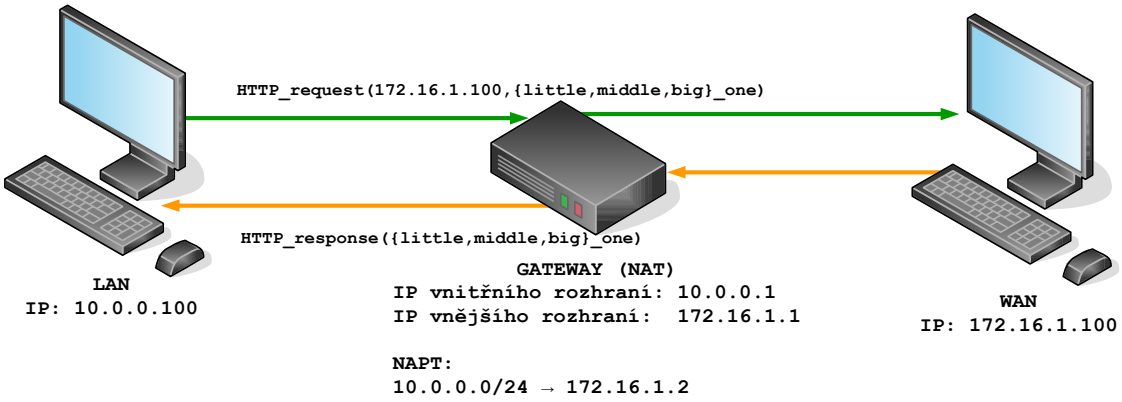
spojení z vnitřní sítě na vnější server, spojení na stanici ve vnitřní síti podle definovaných pravidel. Scénář je úspěšný, pokud fungují správně všechna spojení z vnitřní sítě a lze navázat spojení zvenčí podle neomezeného pravidla. Pro omezené pravidlo by mělo vnější spojení skončit neúspěchem.



Obrázek 7.2: Testovací scénář 2: Restricted vs. Unrestricted pravidla

7.3 Scénář 3: Přenos dat skrze NAT

V tomto scénáři se na rozdíl od prvního scénáře pokusíme navázat déletrvající komunikaci. Jako prostředek nám poslouží stažení datového souboru o přesně známé velikosti. Jako komunikační protokol použijeme HTTP. Nevyžaduje totiž další zpracování a pro FTP je připravený zvláštní scénář. V testu tedy zapojíme klienta a server k zařízení NAT a provedeme přenos souborů. Během testu budeme sledovat úspěšnost přenosu a strukturu provozu během přenosu. Testovací scénář zopakujeme v několika variantách.



Obrázek 7.3: Testovací scénář 3: Přenos dat skrze NAT (HTTP)

7.3.1 Varianta A: Postupný přenos

V tomto případě proběhnou přenosy dat postupně. Vždy tedy zahájíme jedno spojení a počkáme na jeho ukončení. Pokud si zachytíme provoz, tak by měl obsahovat minimální počet chybných paketů nebo jiných chyb během přenosu, jelikož spojení je minimálně vytížené.

7.3.2 Varianta B: Souběžný přenos

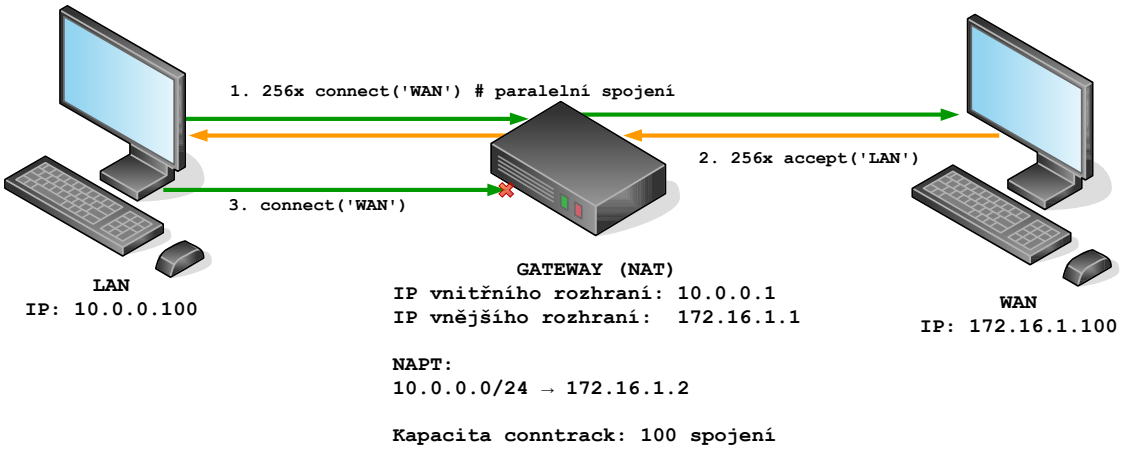
V druhé variantě spustíme všechny přenosy souběžně. Opět nás zajímá úspěšné dokončení. Zároveň provedeme zachycení provozu a budeme sledovat jeho strukturu. V ideálním případě by mělo docházet k minimu opakovaných přenosů a datové toky by měly být konsistentní.

7.3.3 Varianta C: Přenos s emulací reálné linky

V poslední variantě testovacího scénáře upravíme parametry spojení mezi výstupním rozhraním a serverem, abychom simulovali průběh spojení v reálné síti. Pokusíme se tedy vynutit ztrátovost paketů, duplikaci nebo záměnu pořadí. Tyto parametry nejprve ověříme individuálně a poté některé z nich aplikujeme zároveň.

7.4 Scénář 4: Chování při naplněných tabulkách spojení

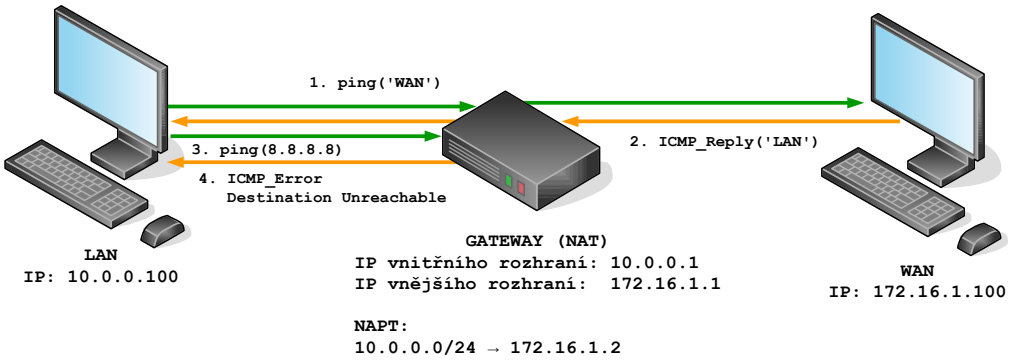
Během své činnosti si NAT ukládá veškerá otevřená spojení, aby mohl správně překládat probíhající provoz a přidělovat dostupné adresy. Kapacita těchto tabulek je samozřejmě omezena a v případě jejich zaplnění je vhodné vědět, jakým způsobem bude NAT reagovat na další spojení. Podle chování můžeme zjistit, co bude NAT dělat během přetížení. Pro test tedy snížíme kapacitu tabulek na relativně nízkou hodnotu, abychom nemuseli generovat velké množství toků. Následně vygenerujeme takový počet spojení, abychom zaplnili tabulky. Poté se pokusíme přidat další spojení. Během testu pak sledujeme stav tabulek, zda nebude některé stávající spojení vyřazeno, což je jedno z možných chování. Druhou možností je, že nové spojení skončí neúspěchem.



Obrázek 7.4: Testovací scénář 4: Chování při naplněných tabulkách spojení

7.5 Scénář 5: Zpracování protokolu ICMP

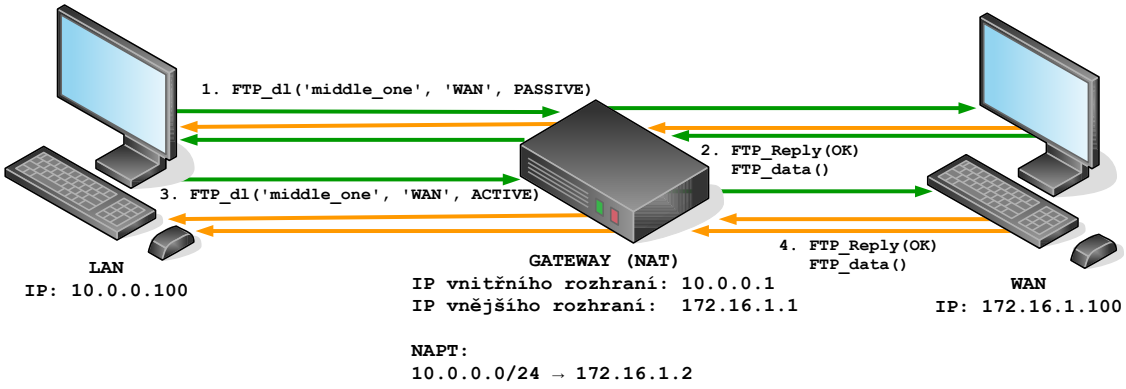
Protokol ICMP slouží pro řízení a diagnostiku spojení na síťové vrstvě. Při průchodu NATem však musí protokol dodržovat určitá pravidla. Některá z nich jsou uvedena v sekci 2.2. V tomto scénáři si tedy ověříme, zda je NAT schopen korektně zpracovat požadavek ICMP a také příslušnou odpověď. Zapojení použijeme stejné jako v předchozích případech. Po nastavení pak vyšleme ICMP echo na server a vyčkáme na úspěšnou odpověď. Poté vyšleme požadavek na nedostupnou adresu. Na něj by měl NAT odpovědět zprávou „Destination Host Unreachable". V obsahu zprávy by pak měly být korektně přeloženy všechny části, tedy jak vnější, tak vnitřní hlavička.



Obrázek 7.5: Testovací scénář 5: Zpracování protokolu ICMP

7.6 Scénář 6: Zpracování protokolu FTP

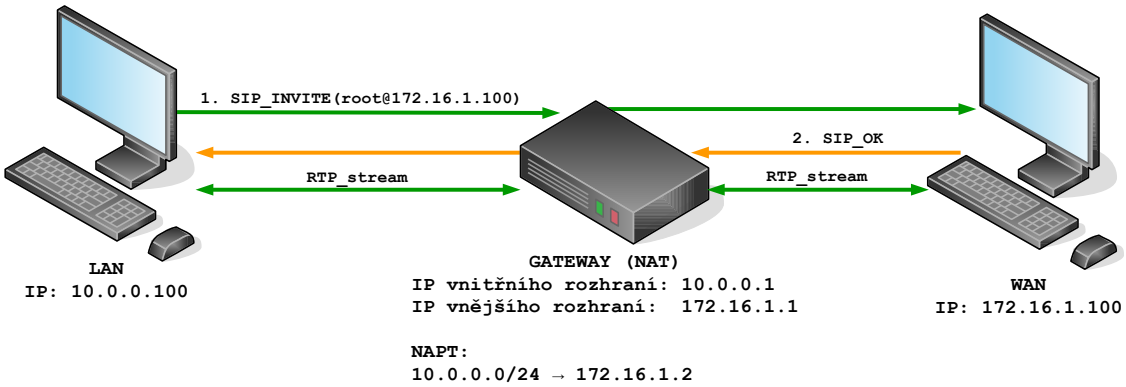
Protokol FTP vyžaduje pro průchod NATem zpracování pomocí ALG. V aktivním režimu totiž otevírá datové spojení server a NAT pro něj nebude mít odpovídající záznam. Bez ALG tak není navázat toto spojení a přenést požadovaná data. Testovací scénář tedy ukáže, zda je zpracování protokolu implementováno na úrovní, že všechna spojení proběhnou v pořádku. Budeme tedy sledovat obsah upravovaných paketů a také sekvenční čísla. Ta musí být upravena v případě, že dojde ke změně délky obsahu paketu, jak je uvedeno v sekci 2.2. Správnost přenosu dat ověříme porovnáním md5 hashe použitých souborů.



Obrázek 7.6: Testovací scénář 6: Zpracování protokolu FTP

7.7 Scénář 7: Zpracování protokolu SIP

Protokol SIP používaný ve VoIP je dalším protokolem, jenž musí být zpracován pomocí ALG. V jeho zprávách jsou totiž obsaženy informace o spojení, kterým budou přenášena hlasová a obrazová data. K takovému spojení nelze explicitně přiřadit pravidlo pro NAT, aniž bychom zpřístupnili část rozsahu portů a potenciálně vytvořili bezpečnostní slabinu. Spojení tedy musí být podobně jako v případě FTP zjištěno z obsahu aplikačního protokolu. V testovacím scénáři se tedy pokusíme navázat SIP spojení mezi stanicemi a pomocí něj přenést předem připravený soubor wav. V úspěšném testu bychom měli vidět korektně zpracované zprávy protokolu SIP a přenos dat protokolem RTP. Data by měla být zachycena na obou stranách komunikace.



Obrázek 7.7: Testovací scénář 7: Zpracování protokolu SIP

7.8 Výkonnostní test porovnávaných platforem

V posledním testu provedeme měření výkonnosti porovnávaných platforem. Zde nás bude zajímat především počet zpracovaných paketů v různých konfiguracích. Pro srovnatelné podmínky bude pro netfilter nastaveno RSS nejdříve pro zpracování jedním jádrem a poté všemi jádry CPU. DPDK nat pak bude spuštěn ve verzi s jedním vláknem NAT worker a poté s využitím všech dostupných vláken. Provoz bude generován pomocí generátoru paketů Spirent Testcenter^{tm} SPT-2000A. Ten bude také zachytávat zpracovaný provoz. Rychlost linky bude postupně zvyšována až na 10 Gb/s.

Kapitola 8

Výsledky testovacích scénářů

V předchozí kapitole byly uvedeny testovací scénáře, které budeme testovat. Tato kapitola je zaměřena na samotné provedení a výsledky těchto scénářů. Ty budou uvedeny především pro vyvíjený nástroj na platformě DPDK. Ve scénářích, kde je potřebné porovnání (zpracování FTP, ICMP apod.), je jako referenční nástroj použit právě netfilter popsaný v kapitole 3. Všechny testy jsou prováděny za pomocí frameworku NTF s výjimkou výkonnostního srovnání. Testovací topologie odpovídá zapojení uvedeném v předchozí kapitole. Parametry jednotlivých stanic jsou uvedeny v příloze B.

8.1 Scénář 1: Navázání spojení skrze NAT

Abychom mohli přejít k ostatním testovacím scénářům, musíme být schopni nejprve provést jednoduché spojení. K tomuto testu využijeme vestavěné funkcionality modulů NTFobserver. Ty při spuštění otevřou port pro naslouchání spojení jak pro TCP, tak pro UDP. Následně vyšleme na tyto porty zprávu a vyčkáme na odpověď. Ta musí přijít do deseti sekund. Samotné spojení zopakujeme vícekrát, především pro UDP. Tím bychom měli minimalizovat riziko ztrát. Ty by však měly být vzhledem k použitému zapojení a vytížení stanic nulové. Test provedeme pouze pro DPDK NAT, jelikož jeho implementace je stále ve vývoji.

```
TEST REPORT ON CONN_TEST
DATE: 16-05-2016 12:43
SUMMARY MARK: 20/20
================================================================================
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using UDP          [ 10/10 ]
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using TCP          [ 10/10 ]
================================================================================
```

Z uvedeného záznamu je vidět, že implementovaný nástroj je schopen správně přeložit přenášené pakety. V případě UDP také vidíme, že nedošlo k žádné ztrátě nebo zahození paketu.

8.2 Scénář 2: Restricted vs. Unrestricted pravidla

V tomto testu se zaměříme na DPDK NAT, a to jeho pravidla pro překlad 1:1. Nástroj umožňuje jejich specifikaci ve dvou variantách, unrestricted a restricted. V první variantě se NAT k provozu zachová jako překladu 1:1 bez doprovodu činnosti firewallu. To znamená, že bude propuštěn jak provoz z privátní sítě, tak provoz přicházející z venčí a směřující na stanici uvedenou v pravidle, resp. na její veřejnou adresu. Činnost pravidla ukazuje následující záznam z provedeného testu. Stanice s adresou `192.168.0.254` je umístěna v simulované privátní sítí, stanice `192.168.0.247` pak v síti věřejné. Záznam ukazuje, že překlad probíhá korektní cestou v obou směrech.

```
TEST REPORT ON RES_UNRES
DATE: 19-05-2016 16:24
SUMMARY MARK: 6/6
================================================================================
192.168.0.247 -- ICMP echo to 172.16.1.10                              [ 1/1 ]
192.168.0.247 -- L4 protcol echo to 172.16.1.10 using UDP              [ 1/1 ]
192.168.0.247 -- L4 protcol echo to 172.16.1.10 using TCP              [ 1/1 ]
192.168.0.254 -- ICMP echo to 172.16.1.100                             [ 1/1 ]
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using UDP             [ 1/1 ]
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using TCP             [ 1/1 ]
================================================================================
```

Pokud specifikujeme stejné pravidlo, jako v předchozím případě, jako restricted, pak bude nástroj provádět kontrolu, zda k příchozímu spojení existuje jeho odchozí protějšek. Jestliže neexistuje, tak je příchozí provoz zahozen. NAT tedy v tomto případě provádí i činnost firewallu, který filtruje provoz na základě stavu spojení. V následujícím záznamu z provedeného testu by tedy měly být všechny příkazy stanice `192.168.0.247` neúspěšné. V opačném směru by komunikace měla proběhnout bez problému.

```
TEST REPORT ON RES_UNRES
DATE: 19-05-2016 16:26
SUMMARY MARK: 6/6
================================================================================
192.168.0.247 -- ICMP echo to 172.16.1.10 (Failure)                    [ 1/1 ]
192.168.0.247 -- L4 protcol echo to 172.16.1.10 using UDP (Failure)    [ 1/1 ]
192.168.0.247 -- L4 protcol echo to 172.16.1.10 using TCP (Failure)    [ 1/1 ]
192.168.0.254 -- ICMP echo to 172.16.1.100                             [ 1/1 ]
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using UDP             [ 1/1 ]
192.168.0.254 -- L4 protcol echo to 172.16.1.100 using TCP             [ 1/1 ]
================================================================================
```

Záznam ukazuje, že komunikace započatá v privátní síti je úspěšná a komunikace přicházející na stejné pravidlo naopak selhává. Podle obou testů lze tedy potvrdit správnou činnosti obou pravidel.

8.3 Scénář 3: Přenos dat skrze NAT

Předchozí testy ukazují, že je možné navázat spojení skrze DPDK NAT. Tato spojení jsou však krátkodobá. V tomto testu tedy vytvoříme komunikaci, která bude trvat déle. K tomu nám poslouží činnost přenesení souboru pomocí protokolu HTTP, který nevyžaduje zpracování pomocí ALG. Podle popisu uvedeném v předchozí kapitole se tedy pokusíme přenést soubory o třech velikostech. Ty jsou postupně pojmenovány „little_one", „middle_one" a „big_one". Jejich velikosti jsou pak 1 MB, 100 MB a 1 GB. Soubory se pak pokusíme přenést nejprve postupně, pak souběžně a nakonec zaneseme do spojení mezi NATem a serverem vyšší chybovost spoje. Ve všech případech požadujeme, aby přenos byl úspěšný. Všechny přenosy pak provedeme desetkrát.

8.3.1 Varianta A: Postupný přenos

V této variantě jsou všechny přenosy prováděny jeden za druhým. V jeden okamžik je tedy aktivní pouze jedno spojení, které musí NAT udržovat. Ze záznamu je zřejmé, že přenos proběhl bez problému pro všechna spojení. Zachycený provoz pak ukazuje velmi nepatrné množství opakovaně přenesených paketů. Ty nejčastěji nesou informaci, že došlo k zaplnění TCP Window.

```
TEST REPORT ON HTTP_ITERATIVE
DATE: 19-05-2016 20:00
SUMMARY MARK: 30/30
================================================================================
192.168.0.254 -- Download of big_one from 172.16.1.100 via http    [ 10/10 ]
192.168.0.254 -- Download of little_one from 172.16.1.100 via http [ 10/10 ]
192.168.0.254 -- Download of middle_one from 172.16.1.100 via http [ 10/10 ]
================================================================================
```

8.3.2 Varianta B: Souběžný přenos

V tomto testu přeneseme stejné soubory, avšak vždy budeme stahovat celou trojici souběžně. NAT tedy musí správně přeložit počátek všech spojení a při ukončení nesmí narušit žádné ze stále probíhajících spojení. Záznam opět ukazuje, že všechny soubory jsou přeneseny. Zachycený provoz je opět s minimem chyb.

```
TEST REPORT ON HTTP_ITERATIVE
DATE: 19-05-2016 20:49
SUMMARY MARK: 30/30
================================================================================
192.168.0.254 -- Download of big_one from 172.16.1.100 via http    [ 10/10 ]
192.168.0.254 -- Download of little_one from 172.16.1.100 via http [ 10/10 ]
192.168.0.254 -- Download of middle_one from 172.16.1.100 via http [ 10/10 ]
================================================================================
```

8.3.3 Varianta C: Přenos s emulací reálné linky

V předchozích variantách předpokládáme, že spojení mezi klientem a serverem funguje bezchybně. Reálná síť však neposkytne ideální podmínky k přenosu. Může totiž dojít ke ztrátě paketu, jeho duplikaci nebo k záměně pořadí mezi pakety. Jmenované chyby se v tomto

testu pokusíme emulovat. Nejprve budeme pozorovat vliv jednotlivých chyb a nakonec zavedeme všechny chyby současně. Celý test provedeme opět se všemi soubory, pouze snížíme počet iteraci na polovinu, tedy 5 přenosů. Přenosy budou provedeny souběžně. Všechny emulované chyby budou nastavovány na rozhraní serveru, jelikož rozhraní NATu nejsou dostupné pro jakoukoli konfiguraci a DPDK neposkytuje nástroje pro emulaci těchto chyb.

Nejprve zavedeme do spojení ztrátovost paketů, kterou nastavíme na 1%. Server tedy zahodí každý stý paket. Tato změna by se měla projevit vyšším počtem opakovaně poslaných paketů, než by se vyskytovalo v čistém přenosu. Soubory by však stále měly být přeneseny. Následující záznam potvrzuje úspěch při přenosu.

```
TEST REPORT ON HTTP_ITERATIVE
DATE: 19-05-2016 21:13
SUMMARY MARK: 15/15
================================================================================
192.168.0.254 -- Download of big_one from 172.16.1.100 via http     [ 5/5 ]
192.168.0.254 -- Download of little_one from 172.16.1.100 via http  [ 5/5 ]
192.168.0.254 -- Download of middle_one from 172.16.1.100 via http  [ 5/5 ]
================================================================================
```

Ztráty paketů se nejvíce projevily při přenosu největšího souboru, kdy se přenos v některých okamžicích na krátkou dobu pozastavil. V žádném přenosu však nebylo spojení přeneseno. V zachyceném provozu se pak objevuje větší počet nesprávně doručených paketů důsledkem chybovosti rozhraní na serveru.

V druhém testu provedeme stejný přenos, avšak pakety budeme duplikovat. Míra duplikace bude nastavena na 5%, tedy každý dvacátý paket bude odeslán ze serveru dvakrát. Tato varianta by neměla výrazně ovlivnit průběh přenosu, avšak v zachycených datech se objeví větší množství paketů s duplicitními údaji v TCP hlavičce.

```
TEST REPORT ON HTTP_ITERATIVE
DATE: 19-05-2016 21:26
SUMMARY MARK: 15/15
================================================================================
192.168.0.254 -- Download of big_one from 172.16.1.100 via http     [ 5/5 ]
192.168.0.254 -- Download of little_one from 172.16.1.100 via http  [ 5/5 ]
192.168.0.254 -- Download of middle_one from 172.16.1.100 via http  [ 5/5 ]
================================================================================
```

Výše uvedený záznam z testu opět ukazuje, že přenosy jsou úspěšné. V zachyceném provozu se pak podle očekávání objevuje množství duplikovaných paketů nebo i jiných chyb.

Posledním parametrem, který budeme testovat, je zpracování paketů v zaměněném pořadí. V celém provozu budeme měnit pořadí u 30% paketů, které zpozdíme o 200 ms. Vlivem tohoto nastavení poklesne rychlost přenosu, avšak nemělo by dojít k přerušení probíhajícího spojení. Záznam opět potvrzuje, že soubory byly přeneseny. V zachycených datech se pak objevuje velké množství nesprávně doručených paketů.

```
TEST REPORT ON HTTP_ITERATIVE
DATE: 19-05-2016 21:36
SUMMARY MARK: 15/15
================================================================================
192.168.0.254 -- Download of big_one from 172.16.1.100 via http     [ 5/5 ]
192.168.0.254 -- Download of little_one from 172.16.1.100 via http  [ 5/5 ]
192.168.0.254 -- Download of middle_one from 172.16.1.100 via http  [ 5/5 ]
================================================================================
```

Nakonec provedeme stejný test se všemi chybami současně. Během testu budeme již nebudeme sledovat zachycený provoz, ale zaměříme se pouze na úspěšnost přenosu. Vzhledem ke kombinaci chyb by takový spoj byl pravděpodobně již odstaven a probíhaly by pokusy o jeho opravu. Zařízení však musí být schopné pracovat správně i v případě taokového scénáře. Záznam opět ukazuje, že soubory jsou staženy úspěšně.

```
TEST REPORT ON HTTP_ITERATIVE
DATE: 20-05-2016 13:05
SUMMARY MARK: 15/15
================================================================================
192.168.0.254 -- Download of big_one from 172.16.1.100 via http     [ 5/5 ]
192.168.0.254 -- Download of little_one from 172.16.1.100 via http  [ 5/5 ]
192.168.0.254 -- Download of middle_one from 172.16.1.100 via http  [ 5/5 ]
================================================================================
```

8.4 Scénář 4: Chování při naplněných tabulkách spojení

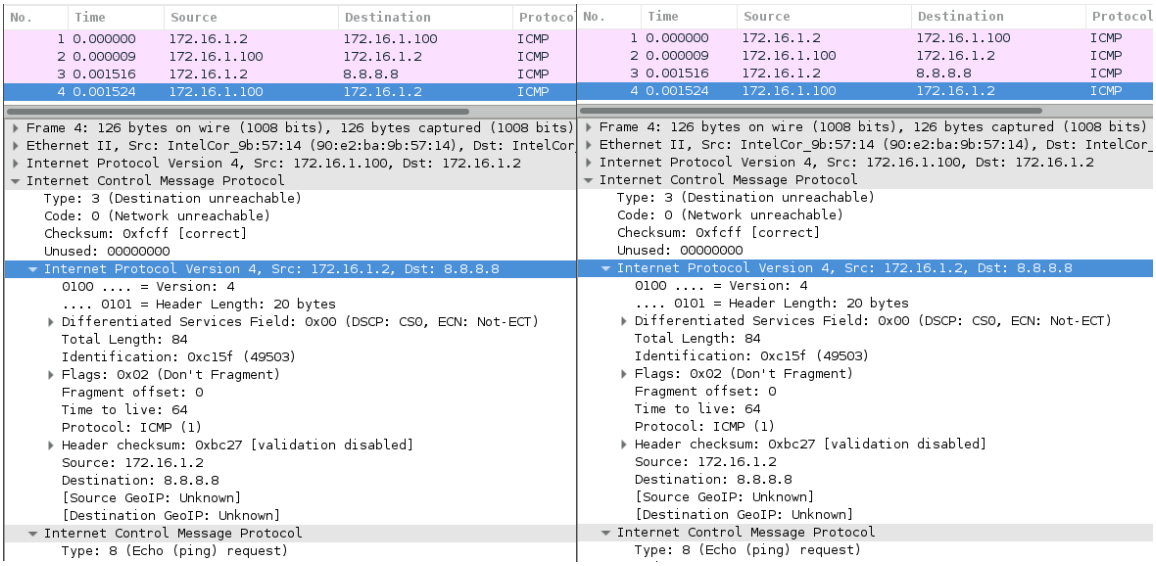
Během své činnosti vytváří každý NAT tabulku aktivních spojení. Její kapacita samozřejmě není neomezená a může dojít k jejímu zaplnění. V této situaci se pak NAT začne chovat k novým spojením v závislosti na implementaci. Modul `nf_conntrack`, jenž používá netfilter, začne v takové situaci další spojení blokovat do doby, než se uvolní kapacita. Stejné chování by měl vykazovat i DPDK NAT, který v tomto scénáři budeme testovat. Abychom při testu nevyčerpali systémové prostředky testovaných stanic, zmenšíme kapacitu tabulky na nejnižší povolenou hodnotu. DPDK NAT povoluje minimálně 256 spojení. Následně vygenerujeme ze stanice stejný počet spojení, abychom zaplnili tabulku a budeme tato spojení udržovat živá. Poté se pokusíme navázat další spojení protokolem ICMP. Jelikož jsou však tabulky NATu zaplněny, ICMP dotaz selže.

```
TEST REPORT ON STRESS_TEST_CONNTRACK
DATE: 19-05-2016 16:15
SUMMARY MARK: 2/2
================================================================================
192.168.0.254 -- Stress test of NAT using 256
                 connections to 172.16.1.100                          [ 1/1 ]
192.168.0.254 -- ICMP echo to 172.16.1.100 (Failure)                  [ 1/1 ]
================================================================================
```

Uvedený záznam ukazuje, že předpokládané chování je správné. DPDK NAT při naplnění kapacity tabulky spojení nepropustí další tok. Jelikož jsou spojení provedena nad protokolem TCP, pak k jejich odebrání dojde až po zachycení paketů s příznakem FIN, případně po vypršení neaktivního timeoutu.

8.5 Scénář 5: Zpracování protokolu ICMP

Při zpracování ICMP nás bude zajímat, zda odpověď projde úspěšně, a také zpracování chybové zprávy. V testu tedy nejprve vyšleme příkazem `ping` zprávu k simulovanému serveru. Ten by měl odpovědět a výsledný paket by měl dorazit. V dalším kroku odešleme požadavek na IP adresu, která není v žádné z testovaných sítí. Netfilter na tuto zprávu odpoví zprávou `Destination Host Unreachable`. DPDK NAT na tuto zprávu neodpoví, ale přepošle ji dále, jelikož on sám není v pozici výchozí brány. Abychom tedy vytvořili chybovou zprávu, povolíme na simulovaném serveru směrování a odebereme výchozí bránu. Po obdržení icmp zprávy pak odpoví chybou `Destination Net Unreachable`, jelikož nezná cestu, na kterou má paket odeslat. V obou případech však musí chybová zpráva dorazit odesilateli dotazu a musí být přeloženy všechny informace, tedy jak vnější IP hlavička, tak hlavička obsažená v chybové zprávě.



Obrázek 8.1: ICMP zpracované DPDK NATem: vnitřní rozhraní (vlevo), vnější rozhraní (vpravo)

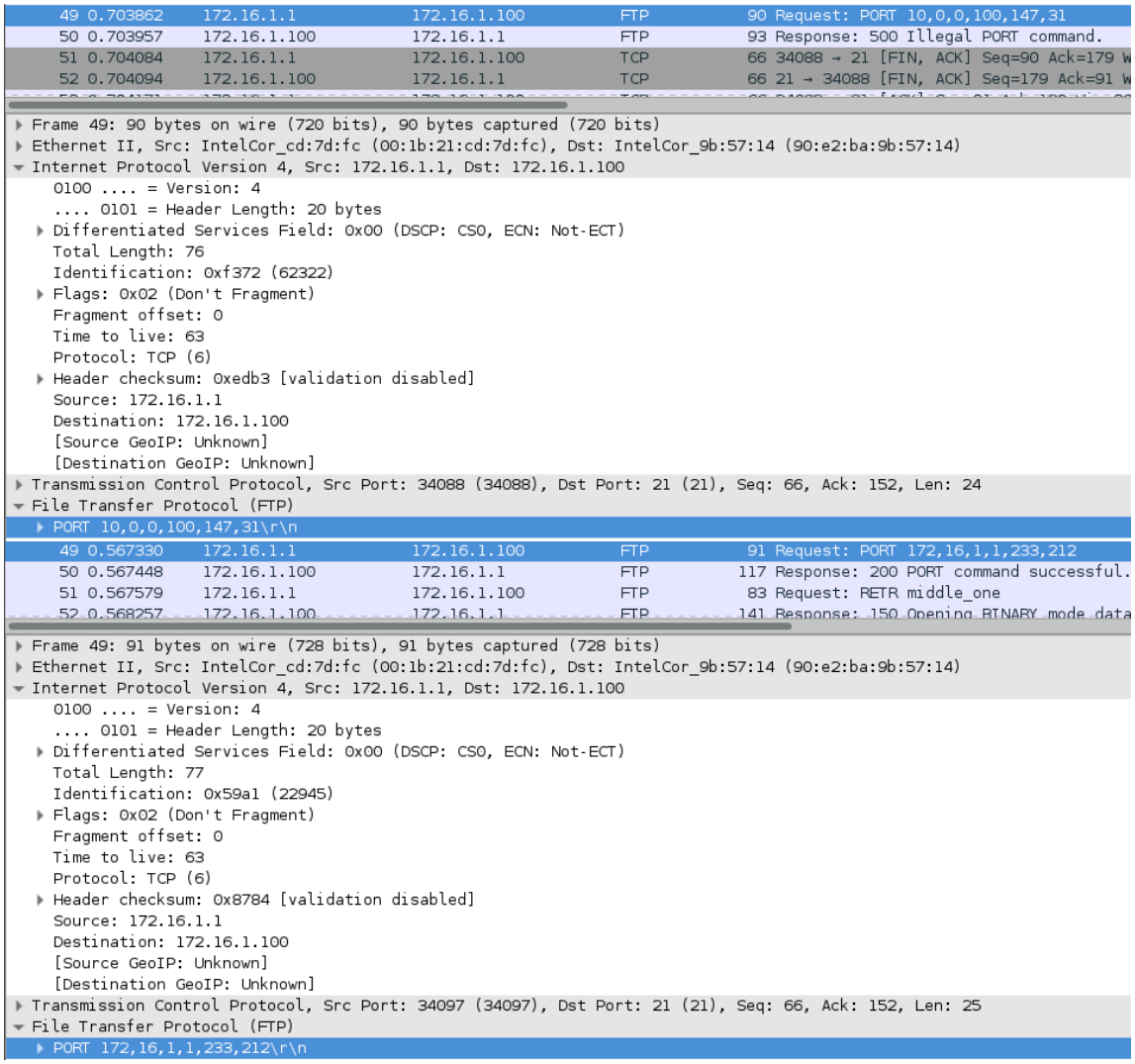
Na obrázku 8.1 vidíme provoz zachycený během testu DPDK NATu. Zvýrazněný paket ukazuje vnitřní hlavičku chybové zprávy jak na straně odesilatele dotazu (vlevo), tak na straně příjemce dotazu (vpravo), jenž zprávu generuje. Z obrázku je zřejmé, že pakety jsou správně přeloženy, jelikož provoz je kompletní a v chybové zprávě jsou přeloženy všechny informace.

8.6 Scénář 6: Zpracování protokolu FTP

V testu zpracování FTP nás především zajímá úspěšnost spojení. Pokud tedy NAT implementuje ALG a podporuje FTP, pak by spojení mělo být úspěšné jak pro pasivní tak aktivní režim. Pro ilustraci si nejprve ukážeme provoz skrze NAT, který neumožňuje zpracování protokolu FTP. Ukázka je vytvořena pomocí nástroje netfilter.

V zachyceném provozu uvidíme nejprve pasivní režim, jehož spojení proběhne korektně. Nedochází totiž k přenosu příkazu `PORT`, který by obsahoval informace o datovém toku. Tento tok je v pasivním režimu otevírán ze strany klienta a projde tedy skrze NAT bez potíží. Jiná situace nastane u aktivního režimu. U něj pozorujeme, že NAT správně přeložil IP hlavičku paketů, ale data přenášená s příkazem PORT odkazují na stanici v privátní síti. FTP server však očekává spojení od jiného klienta a příkaz `PORT` je zamítnut.

Nyní test zopakujeme se stejnými parametry, avšak zavedeme do jádra modul `nf_nat_ftp`. Ten způsobí, že netfilter bude zpracovávat data protokolu FTP. Nyní by tedy měla proběhnout všechna spojení, tedy jak pasivní, tak aktivní. Obrázek 8.2 ukazuje příkaz `PORT` v přeložené a nepřeložené verzi.



Obrázek 8.2: Porovnání příkazů FTP bez ALG (nahoře) a s ALG (dole)

Nyní stejný test provedeme s nástrojem pro DPDK. Ten používá podobnou metodu zpracování, kdy modul FTP může být přidán za běhu. Spojení opět proběhnou úspěšně, jak ukazuje záznam. Zajímavý je rozdíl ve výběru portů. Zatímco netfilter zachovává port vybraný stanicí v privátní síti, DPDK NAT alokuje porty vždy od hodnoty 1024. To však může být způsobeno tím, že netfilter běží v systému neustále. DPDK NAT je před testem vždy restartován. Další příčinou tohoto rozdílu je sémantika pravidla pro DPDK NAT.

```
TEST REPORT ON FTP_ITERATIVE
DATE: 20-05-2016 16:58
SUMMARY MARK: 2/2
================================================================================
192.168.0.254 -- Download of middle_one from 172.16.1.100
                 via passive ftp                                        [ 1/1 ]
192.168.0.254 -- Download of middle_one from 172.16.1.100
                 via active ftp                                         [ 1/1 ]
================================================================================
```

8.7 Scénář 7: Zpracování protokolu SIP

Při testování protokolu SIP požadujeme, aby došlo nejen k překladu vlastních paketů, ale také ke zpracování aplikačních dat. V nich totiž tento protokol přenáší informace o spojení použitém k přenosu audiovizuálních dat. Ty mohou bez zpracování způsobit, že se spojení nenaváže vůbec, nebo pakety budou vyslány neočekávanou cestou. NAT by tedy měl zpracovávat zprávy, které přenášejí tyto informace, aby znal parametry datového spojení. V případě SIP se jedná o zprávy typu INVITE. Pro provedení testu použijeme klienta `linphonec` v konzolové variantě.

Netfilter řeší zpracování pomocí modulu `nf_nat_sip`. Bez něj se přenos provede poněkud neobvyklým způsobem. Datové spojení je během testu navázáno, avšak pakety neproudí očekávanou cestou. Místo průchodu přes zařízení NAT, jsou data směrována skrze konfigurační rozhraní. To je způsobeno použitým zapojením, kde zařízení jsou sice propojena přímo, ale konfigurační rozhraní jsou připojena do přepínače. V zachycených datech pak uvidíme pouze SIP komunikaci, která zároveň není úplná. V jejím obsahu pak nebudou přeloženy potřebné položky, především položka Contact v hlavičce protokolu SDP a port pro ustavení RTP spojení. Níže můžete vidět obsah nepřeloženého SIP paketu.

```
INVITE sip:root@10.0.0.100 SIP/2.0
Via: SIP/2.0/UDP 10.0.0.100:5060;rport;branch=z9hG4bK1156542842
From: <sip:root@10.0.0.100:5060>;tag=112021105
To: <sip:root@172.16.1.100>
Call-ID: 934136126
CSeq: 20 INVITE
Contact: <sip:root@10.0.0.100:5060>
Content-Type: application/sdp

...

v=0
o=root 1016 1168 IN IP4 10.0.0.100
s=Talk
c=IN IP4 10.0.0.100
t=0 0
m=audio 7078 RTP/AVP 124 111 110 0 8 101

...
```

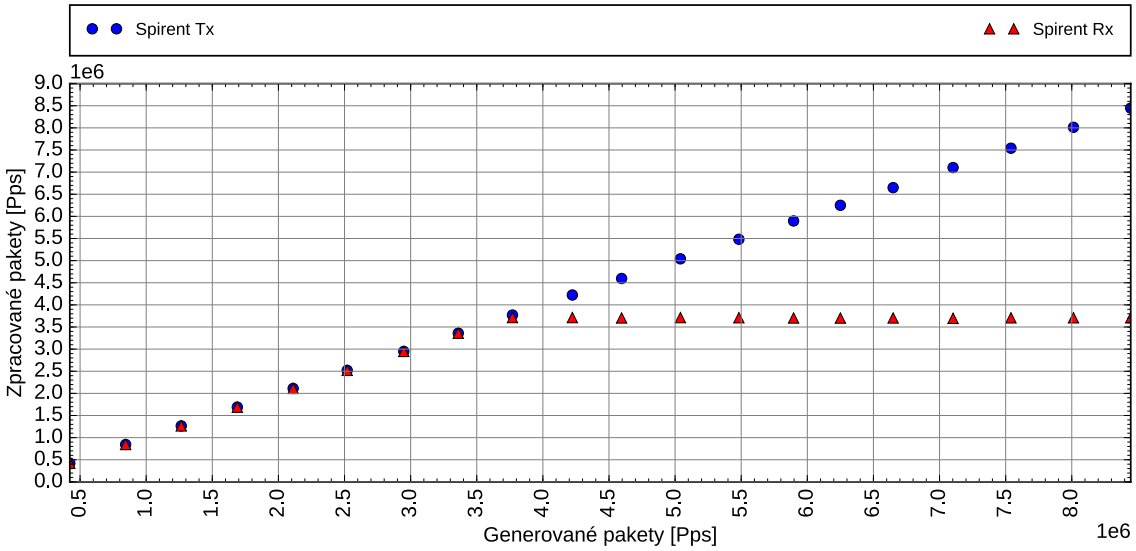
DPDK NAT zpracovává protokol také pomocí přídavného modulu. Ten bude zaveden při startu nástroje. V současnosti modul podporuje pouze odchozí hovory. zaměříme se tedy na funkčnost této vlastnosti. Požadujeme tedy, aby všechna data proudila správnými rozhraními a komunikace SIP byla korektně přeložena.

```
# Paket přeložený pomocí netfilter

INVITE sip:root@172.16.1.100 SIP/2.0
Via: SIP/2.0/UDP 172.16.1.1:5060;rport;branch=z9hG4bK1156542842
From: <sip:root@172.16.1.1:5060>;tag=112021105
To: <sip:root@172.16.1.100>
Call-ID: 934136126
CSeq: 20 INVITE
Contact: <sip:root@172.16.1.1:5060>

...

v=0
o=root 1016 1168 IN IP4 172.16.1.1
s=Talk
c=IN IP4 172.16.1.1
m=audio 7078 RTP/AVP 124 111 110 0 8 101
...


# Paket přeložený pomocí DPDK

INVITE sip:root@172.16.1.100 SIP/2.0
Via: SIP/2.0/UDP 172.16.1.2:1024;rport;branch=z9hG4bK1234182214
From: <sip:root@10.0.0.100>;tag=275576157
To: <sip:root@172.16.1.100>
Call-ID: 399500218
CSeq: 20 INVITE
Contact: <sip:root@172.16.1.2>

...

v=0
o=root 1843 3127 IN IP4 10.0.0.100
s=Talk
c=IN IP4 172.16.1.2
t=0 0
m=audio 1025 RTP/AVP 124 111 110 0 8 101

...
```

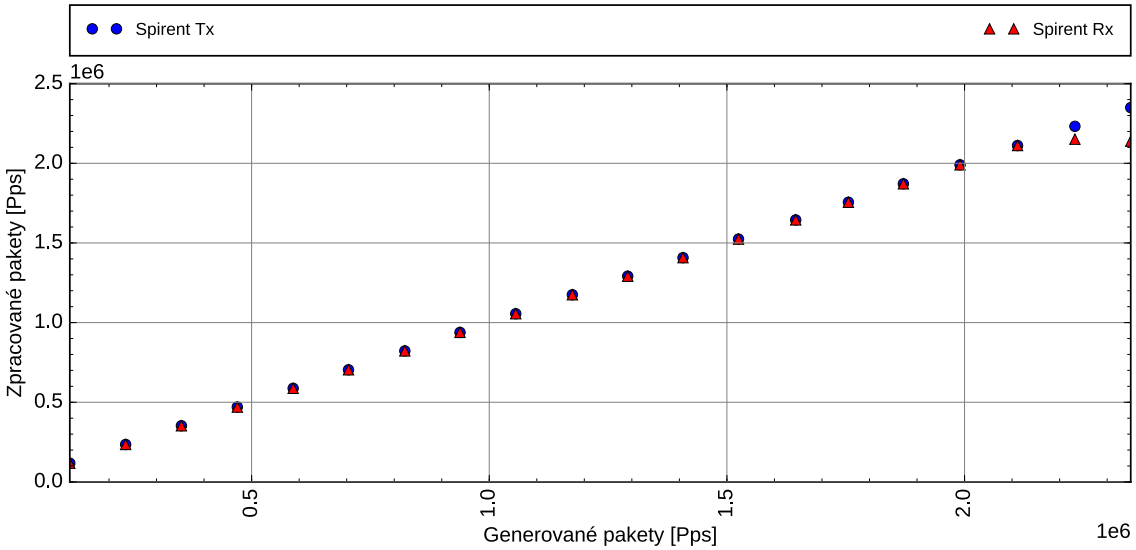
Výše uvedené pakety ukazují funkci překladu pro netfilter a dpdk. Všimněme si rozdílů v přeložených paketech, kde DPDK NAT nepřekládá pole `From`. V klientovi se tedy zobrazí příchozí hovor jdoucí z privátní adresy. Spojení však bude probíhat korektně skrze překladovou stanici. Pro ustavení komunikace je totiž používána položka `Contact`. Další rozdíl je vidět v přiřazení portů. Netfilter zachovává porty přiřazené pro SIP spojení, kdežto DPDK NAT alokuje porty vlastní. Tento rozdíl je způsoben specifikací pravidla pro DPDK NAT a také způsobem alokace dvojice adresa-port.

8.8 Výkonnostní test porovnávaných platforem

Ve výkonnostním testu budeme srovnávat maximální možnou propustnost obou nástrojů. Jako testovací data použijeme provoz vygenerovaný zařízením Spirent Testcenter^{tm} SPT-2000A. Ten bude vytvářet provoz sestávající z 10000 toků, abychom bylo možné zátěž rozložit mezi jádra. S netfilter je test proveden s nastavením RSS tak, aby každé jádro CPU zpracovávalo jednu frontu síťové karty. DPDK NAT pak bude testován v konfiguraci s jedním aktivním jádrem, tedy jedno jádro CPU zároveň distribuuje a překládá. Následně rozložíme distribuci a překlad mezi dvě jádra. Nakonec budeme testovat konfiguraci se dvěma překladovými jádry.

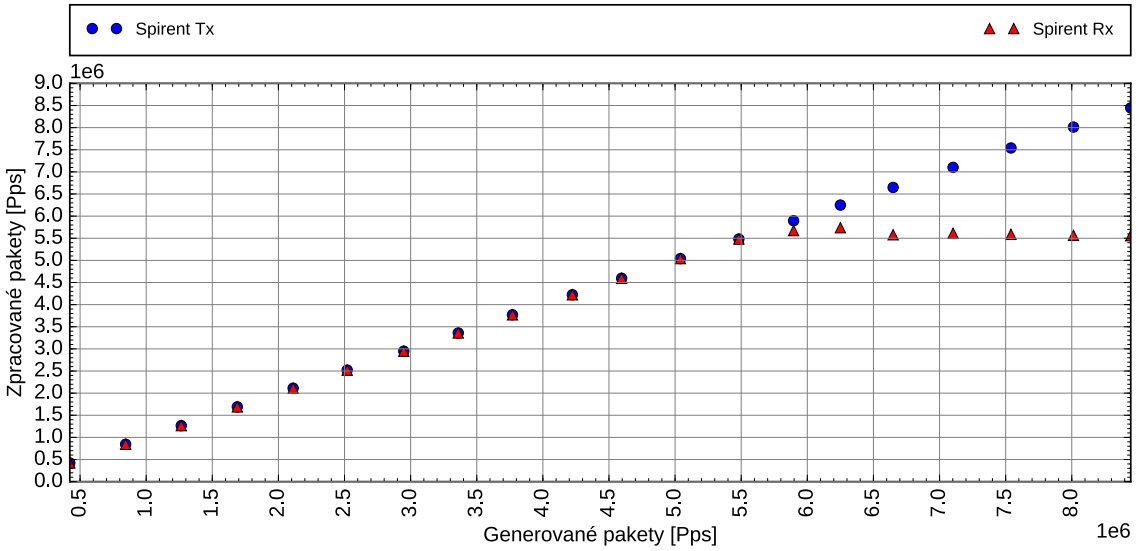


Obrázek 8.3: Výkonnost zpracování paketů pomocí netfilter (délka paketu 128 B)

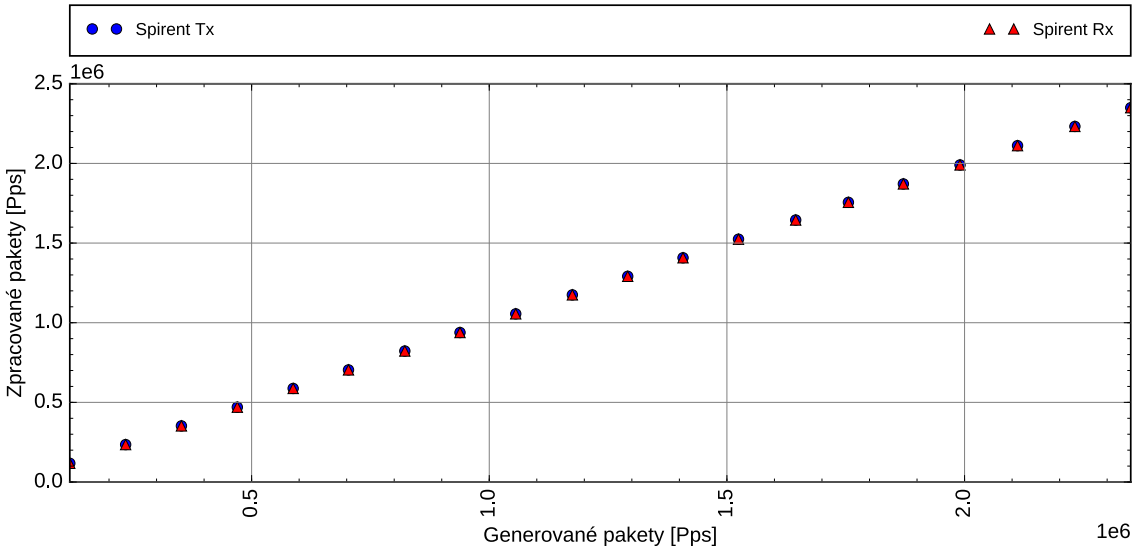


Obrázek 8.4: Výkonnost zpracování paketů pomocí netfilter (délka paketu 512 B)

Na grafech 8.3 a 8.4 je zobrazena výkonnost zpracování paketů pomocí netfilter pro velikosti 128 a 512 bytů. Modré body ukazují pakety vycházející z generátoru, červené pakety, které byly zpracovány NATem a zachyceny generátorem. Při popsané konfiguraci je netfilter schopen zpracovat cca 3,7 milionu paketů za sekundu při délce 128 bytů. Během testu jsou pak všechna jádra CPU vytížena na maximum. Nyní provedeme stejný test pro DPDK NAT.



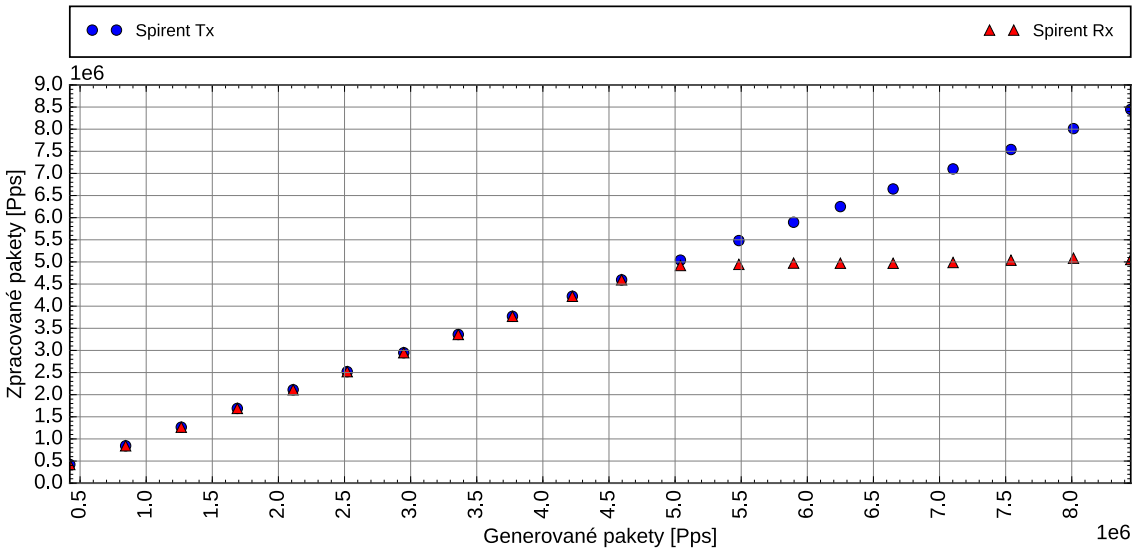
Obrázek 8.5: Výkonnost zpracování paketů pomocí DPDK (délka paketu 128 B, 1 jádro)



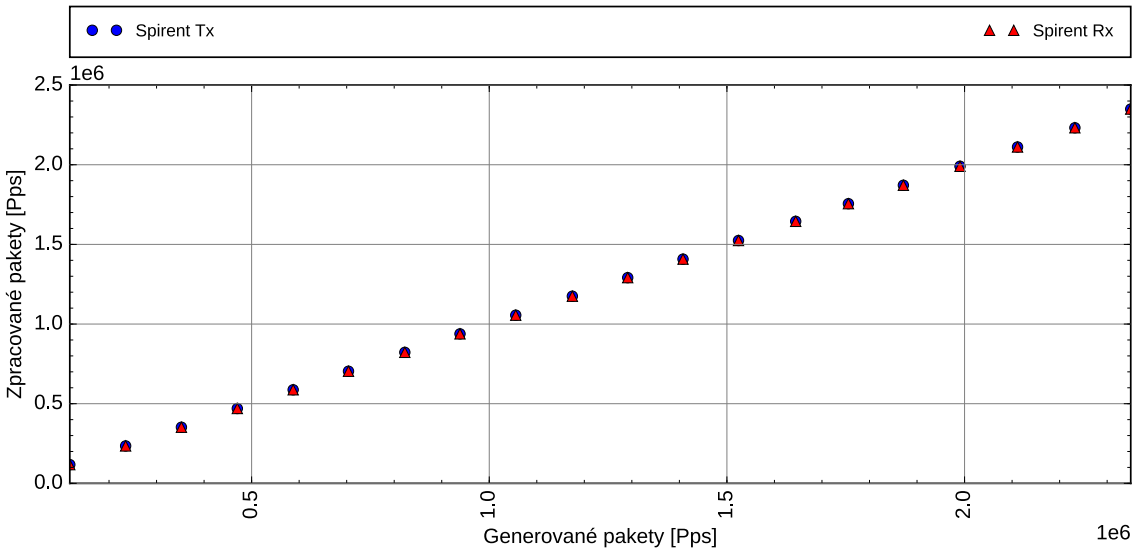
Obrázek 8.6: Výkonnost zpracování paketů pomocí DPDK (délka paketu 512 B, 1 jádro)

Jak ukazují grafy 8.5 a 8.6, tak již v jednojádrové konfiguraci ukazuje DPDK NAT výrazně vyšší výkon při zpracování. Ten je způsoben především díky tomu, že paket nemusí procházet některými moduly, kterými by procházel při zpracování netfilterem. Dalším přínosem je minimální počet zámků, jelikož struktury pro ukládáním paketů jsou v DPDK

implementovány bezzámkově. Nyní test provedeme v konfiguraci, kdy distribuci a zpracování budou provádět dvě jádra. Obě jádra jsou lokalizována na stejném uzlu NUMA[1], a také na nezávislých fyzických jádrech, aby nedocházelo k hyperthreadingu.



Obrázek 8.7: Výkonnost zpracování paketů pomocí DPDK (délka paketu 128 B, 1 jádro distribuce, 1 jádro zpracování)

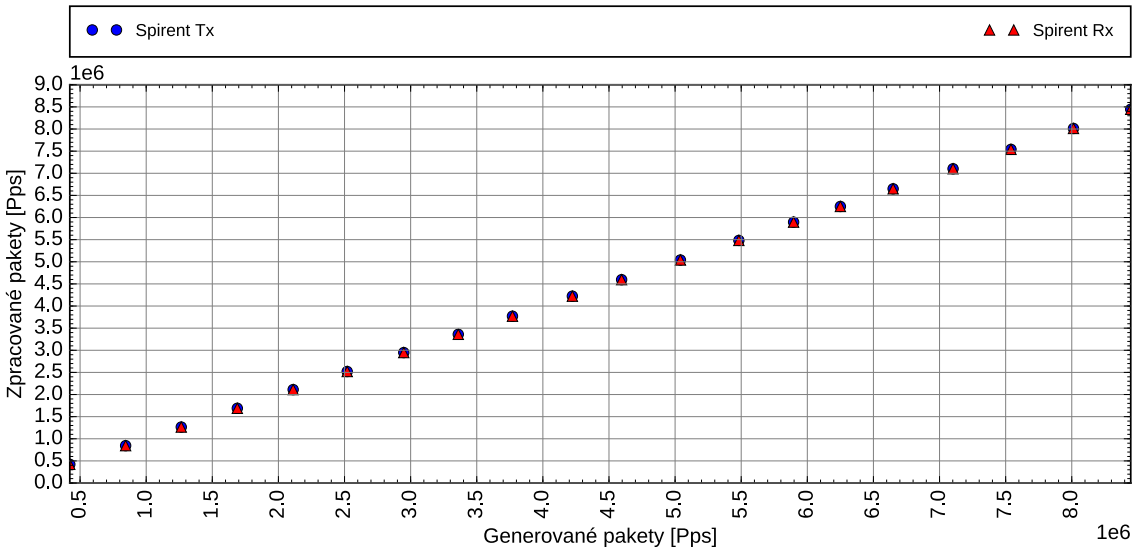


Obrázek 8.8: Výkonnost zpracování paketů pomocí DPDK (délka paketu 512 B, 1 jádro distribuce, 1 jádro zpracování)

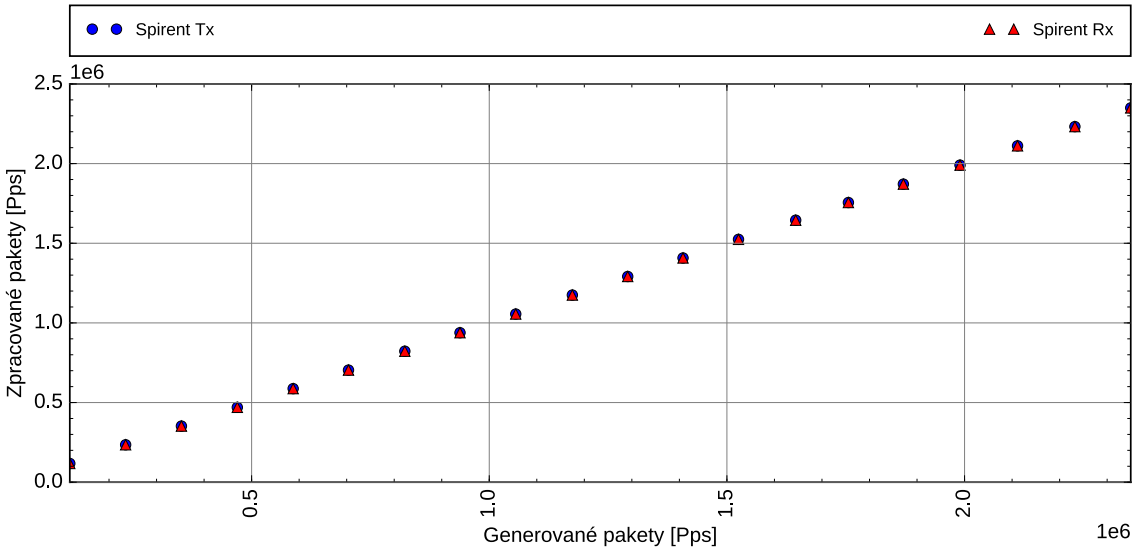
V této konfiguraci je vidět velmi podobný výkon jako v čistě jednojádrové konfiguraci. Celkový výkon je však o trochu nižší. Příčinou tohoto drobného propadu je nutnost synchronizace mezi jádry.

[1]Non-Uniform Memory Access

Nakonec provedeme test se dvěma zpracovávajícími jádry. Zátěž by měla být mezi nimi rozložena přibližně v poměru 50:50.



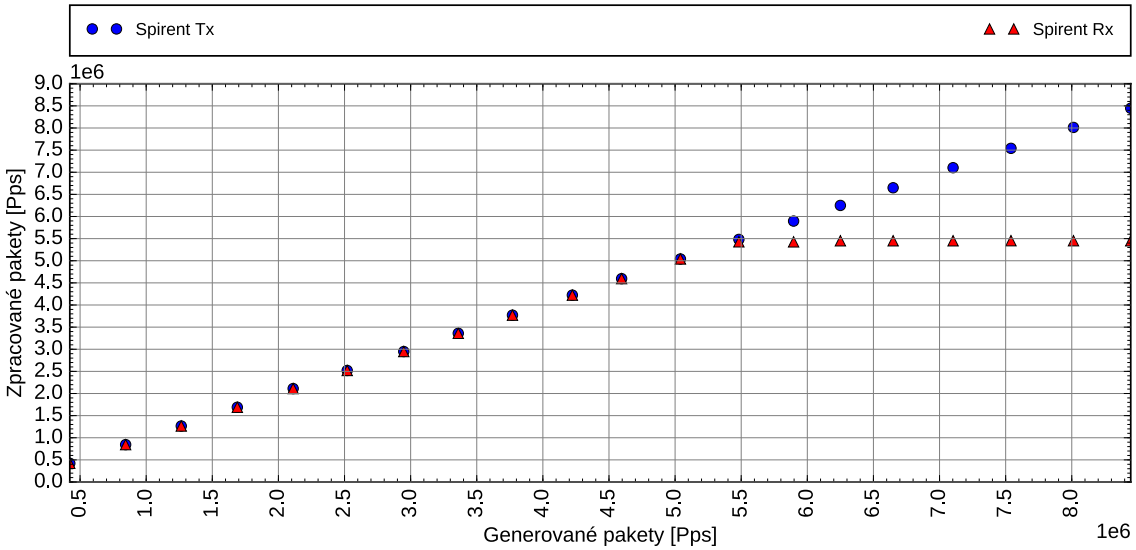
Obrázek 8.9: Výkonnost zpracování paketů pomocí DPDK (délka paketu 128 B, 1 jádro distribuce, 2 jádra zpracování)



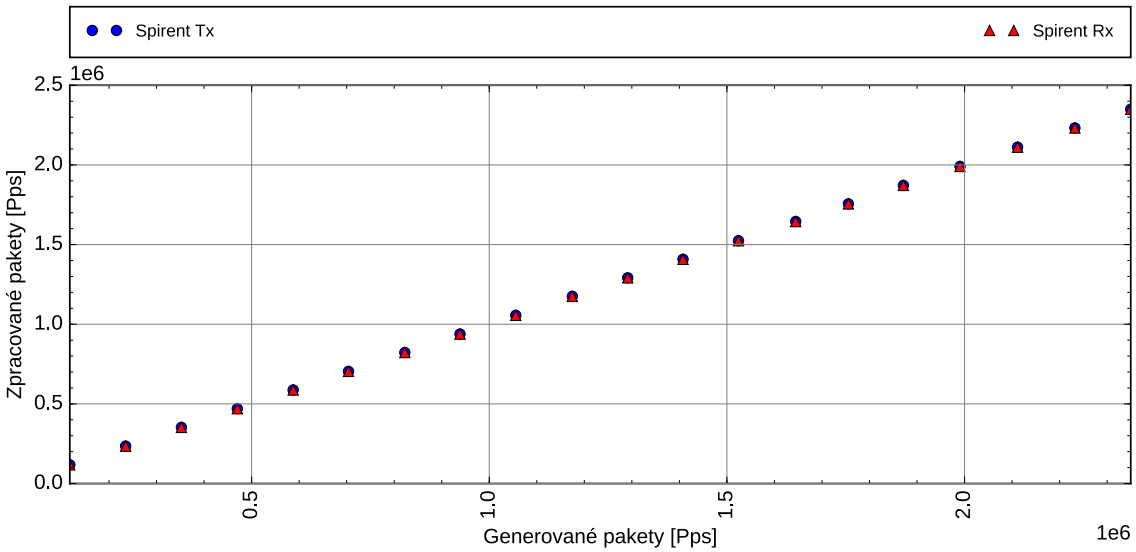
Obrázek 8.10: Výkonnost zpracování paketů pomocí DPDK (délka paketu 512 B, 1 jádro distribuce, 2 jádra zpracování)

Po přidání dalšího jádra vidíme v grafech 8.9 a 8.10, že rychlost zpracování dosahuje rychlosti linky. DPDK NAT je tedy schopen v konfiguraci se dvěma jádry pro překlad zpracovat provoz na rychlosti 10 Gb/s v plném rozsahu bez ztráty paketu.

Při předchozích testech jsme vlákna mapovali na samostatná fyzická jádra. V posledním testu se tedy podíváme na výkon NATu při využití všech dostupných jader. Použitý procesor poskytuje šest fyzických jader s podporou Hyperthreadingu, tedy až 12 logických výpočetních jednotek.



Obrázek 8.11: Výkonnost zpracování paketů pomocí DPDK (délka paketu 128 B, všechna dostupná jádra)



Obrázek 8.12: Výkonnost zpracování paketů pomocí DPDK (délka paketu 512 B, všechna dostupná jádra)

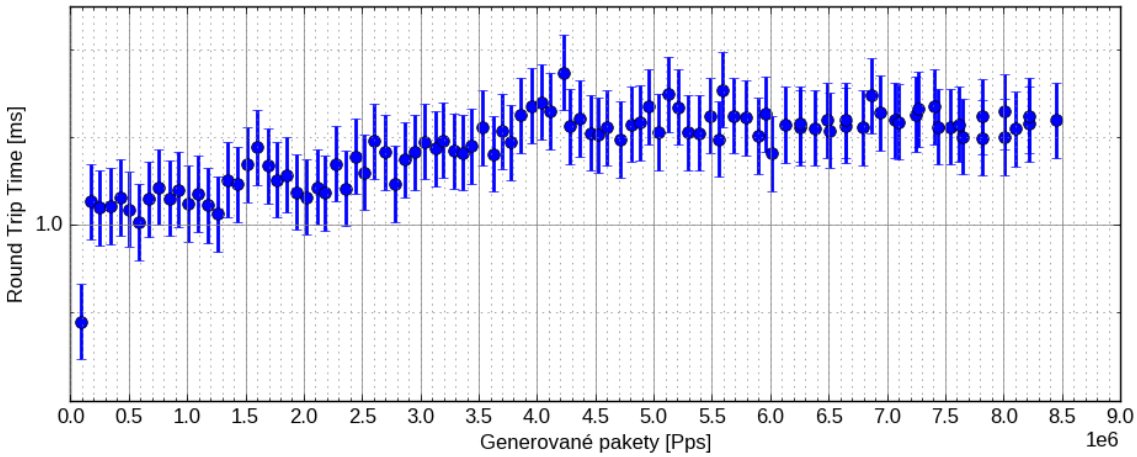
Z grafů 8.11 a 8.12 je vidět, že výkonnost zpracování velmi výrazně klesla. To způsobuje fakt, že všechna jádra provádějí shodnou činnost s jinými daty. Jelikož jsou používaná data uložena na různých místech v paměti, dochází k velmi častému promazání pamětí cache, které jsou sdíleny mezi logickými jádry. Jednotlivá vlákna jsou také při maximální rychlosti

stále zatížena zpracováním paketů a neposkytují tedy mnoho okamžiků, kdy by bylo možné činnost vláken „prokládat".

Na závěr testů propustnosti uvedeme tabulku se všemi testovanými případy a jejich výsledky. V ní jsou uvedeny jak celkové propustnosti v milionech paketů za sekudu, tak podíl zpracovaného provozu oproti generovanému.

Překladový nástroj	**Zpracované pakety 128 B**		**Zpracované pakety 512 B**	
	[Mpkt/s]	**[%]**	**[Mpkt/s]**	**[%]**
netfilter	3,70	43,8	2,15	91,8
DPDK (1 jádro)	5,74	68,0	2,34	100,0
DPDK (1 + 1 jádro)	5,08	60,1	2,34	100,0
DPDK (1 + 2 jádra)	8,44	100,0	2,34	100,0
DPDK (1 + 11 jader HT)	5,45	64,5	2,34	100,0
Generované pakety	8,44	100,0	2,34	100,0

Metodika uvedená v kapitole 5 popisuje mimo testu propustnosti také test odezvy serveru pomocí RTT. Jak je ale patrné z grafu 8.13, tak rychlost odezvy se díky použitému zapojení pohybuje v jednotkách milisekund i menších. Z pohledu uživatele tedy samotné zařízení nemá výrazný vliv na rychlost odezvy, pokud je nastavení zařízení korektní. Pokud bychom chtěli zvýšit odezvu, museli bychom uměle nastavit zpoždění na některém rozhraní, nebo mezi NAT a server vložit další zařízení.



Obrázek 8.13: Výkonnost zpracování paketů pomocí DPDK (délka paketu 512 B, všechna dostupná jádra)

Z hlediska využití hardwarových prostředků je DPDK NAT výrazně náročnější na paměť než netfilter. DPDK NAT totiž pro veškeré alokace využívá tzv. hugepages. Ty zajišťují, že DPDK NAT může využít velký paměťový prostor, který bude zároveň souvislý. Vyhrazené stránky NAT zabere okamžitě po spuštění. Pro systém se tedy tato paměť jeví jako obsazená. Pokud budeme sledovat vytížení procesoru, zjistíme, že jádra používaná DPDK NATem jsou neustále plně vytížená. To je způsobeno tím, že DPDK nepracuje s přerušeními, ale neustále se dotazuje rozhraní, zda neobsahuje nějaký paket. Netfilter dosáhne maximálního vytížení pouze v případě, že rozhraní přestanou stíhat zpracovávat provoz.

Kapitola 9

Plánovaná a možná rozšíření

Nástroj navržený a implementovaný v kapitole 6 je schopen provést všechny testovací scénáře. Rozsah jeho možností je však v současnosti velmi omezený- Můžeme tedy provádět pouze relativně základní úkony. Pokud však potřebujeme podrobnější testování, musíme použít jiné prostředky. V této kapitole se podíváme na některá možná rozšíření, které by mohly vylepšit práci s nástrojem a rozvinout rozsah jeho schopností.

9.1 Vylepšení autokonfigurace zařízení

V současnosti je framework schopen automaticky konfigurovat pouze koncové stanice. Konfigurace NATu a případných dalších zařízení tedy musí být provedena ručně. Díky tomu může být problém, že test neproběhne správně, jelikož konfigurace některého zařízení je chybná. V následujícím vývoji je plánována funkce automatické konfigurace těchto zařízení s ohledem na nástroj, jenž je pro tyto konfigurace určen. Primárně bude tato funkce dostupná pro zařízení založená na systému Linux nebo BSD, případně pro nástroje, které budou poskytovat možnost konfigurace za běhu. Samozřejmě je tato funkcionalita omezena možností spustit modul NTFobserver na daném zařízení.

9.2 Ověření popisu a zapojení topologie

Skript popisující použitou topologii definuje použitá zařízení, jejich rozhraní a propojení mezi nimi. Tato konfigurace je pak aplikována bez ohledu na její správnost. Pokud je tedy chybně uvedená některá adresa nebo dojde k duplicitě adres, bude tato skutečnost rozpoznána až během testu. V následujících verzích bude před započetím testu přidána kontrola, zda popsaná topologie neobsahuje konflikty adres nebo zda všechna zařízení náleží do správných sítí. Budou také kontrolována případná pravidla pro NAT, zda nejsou zbytečná nebo chybně specifikovaná.

S touto funkcí souvisí i kontrola fyzického zapojení. Zařízení jsou před testem nakonfigurována a spuštěný test předpokládá funkčnost všech zapojení. V následujících verzích tedy bude přidána kontrola topologie na úrovní spojení point-to-point. V případě chybného spoje bude test přerušen a framework oznámí, které dvojice zařízení nejsou vzájemně dostupné.

9.3 Automatická analýza zachycených dat

Framework je v současnosti schopen vyhodnotit testovací scénář pouze na základě úspěšnosti celého úkonu. Pokud tedy požadujeme podrobnější analýzu provozu, musí být zachycená data vyhodnocena ručně. V dalších verzích by měl být framework schopen analýzy zachycených dat. Bude tedy například vytvářet základní statistiky jako jsou množství přenesených dat či počty chybně odeslaných paketů. Dále je jednou z plánovaných funkcí přidání analýzy proběhnutých překladů. NAT aplikuje překladová pravidla podle určitých priorit v případě existence pravidel 1:1 a M:N. Lze předpovědět, jaká pravidla by měla být použita, a pro které toky. Pokud bude možné získat obsah tabulek testovaného NATu, pak můžeme vytvořit očekávané překlady a tyto pak hledat v zachyceném provozu. Pokud budou překlady odpovídat, pak NAT vykonává svou činnost správně.

9.4 Podpora generátorů paketů a rozšíření na výkonnostní testy

Současná verze frameworku je zaměřena spíše na funkční testy. Sleduje tedy úspěšnost datových spojení a jejich chování. Výkonnostní testy tedy musí být prováděny jinou metodou. V budoucích verzích je plánováno přidat podporu i pro výkonnostní testy. Pro jejich spuštění tedy bude nutné přidat podporu pro vysokorychlostní generátory provozu jako jsou např. zařízení Spirent^{tm}.

Kapitola 10

Závěr

V práci byla popsána činnost mechanismu pro překlad síťových adres a jeho aplikace v praxi. Byly popsány způsoby překladu samotných paketů i práce s aplikačními protokoly. Jako případové studie byly vybrány implementace netfilter pro referenční řešení a dále nově vyvíjený nástroj nad frameworkem Intel DPDK. Dále jsou v práci popsány metodiky testování síťových zařízení a jejich aplikace na překladové nástroje.

V rámci práce byl dále implementován framework NTF, který je následně použit jako testovací nástroj pro vyvíjený DPDK NAT. Ten byl společně s netfilter testovaný jak z pohledu výkonnosti zpracování, tak z pohledu funkcionality. Z výsledků testů v kapitole 7 je zřejmé, že nově vznikající NAT má vysoký výkonnostní potenciál a je také schopen základní práce s aplikačními protokoly. Framework NTF bude podle potřeby rozšiřován o další funkce a bude využíván k dalšímu testování během vývoje DPDK NATu či jiného podobného nástroje.

Literatura

[1] IEEE Standard for Local and Metropolitan Area Networks – Station and Media Access Control Connectivity Discovery. 2009.

[2] *DPDK: Data Plane Development Kit*. Duben 2016.
URL http://dpdk.org/doc/guides/

[3] Baker, F.: Requirements for IP Version 4 Routers. RFC 1812 (Proposed Standard), Červen 1995, updated by RFCs 2644, 6633.
URL http://www.ietf.org/rfc/rfc1812.txt

[4] Bradner, S.; Dubray, K.; McQuaid, J.; aj.: Applicability Statement for RFC 2544: Use on Production Networks Considered Harmful. RFC 6815 (Informational), Listopad 2012.
URL http://www.ietf.org/rfc/rfc6815.txt

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

[6] Farinacci, D.; Li, T.; Hanks, S.; aj.: Generic Routing Encapsulation (GRE). RFC 2784 (Proposed Standard), Březen 2000, updated by RFC 2890.
URL http://www.ietf.org/rfc/rfc2784.txt

[7] Frankel, S.; Krishnan, S.: IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap. RFC 6071 (Informational), Únor 2011.
URL http://www.ietf.org/rfc/rfc6071.txt

[8] Hanks, S.; Li, T.; Farinacci, D.; aj.: Generic Routing Encapsulation (GRE). RFC 1701 (Informational), Říjen 1994.
URL http://www.ietf.org/rfc/rfc1701.txt

[9] Hanks, S.; Li, T.; Farinacci, D.; aj.: Generic Routing Encapsulation over IPv4 networks. RFC 1702 (Informational), Říjen 1994.
URL http://www.ietf.org/rfc/rfc1702.txt

[10] Holdrege, M.; Srisuresh, P.: Protocol Complications with the IP Network Address Translator. RFC 3027 (Informational), Leden 2001.
URL http://www.ietf.org/rfc/rfc3027.txt

[11] Huston, G.: Anatomy: A Look Inside Network Address Translators. *The Internet Protocol Journal*, ročník 7, č. 3, Září 2004.
URL http://www.cisco.com/c/dam/en_us/about/ac123/ac147/archived_issues/ipj_7-3/ipj_7-3.pdf

[12] Kent, S.; Seo, K.: Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Prosinec 2005, updated by RFCs 6040, 7619.
URL http://www.ietf.org/rfc/rfc4301.txt

[13] Postel, J.: Internet Protocol. RFC 791 (INTERNET STANDARD), Září 1981, updated by RFCs 1349, 2474, 6864.
URL http://www.ietf.org/rfc/rfc791.txt

[14] Postel, J.; Reynolds, J.: File Transfer Protocol. RFC 959 (INTERNET STANDARD), Říjen 1985, updated by RFCs 2228, 2640, 2773, 3659, 5797, 7151.
URL http://www.ietf.org/rfc/rfc959.txt

[15] Rosenberg, J.; Schulzrinne, H.; Camarillo, G.; aj.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), Červen 2002, updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141, 6665, 6878, 7462, 7463.
URL http://www.ietf.org/rfc/rfc3261.txt

[16] Rosenberg, J.; Weinberger, J.; Huitema, C.; aj.: STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), Březen 2003, obsoleted by RFC 5389.
URL http://www.ietf.org/rfc/rfc3489.txt

[17] Senie, D.: Network Address Translator (NAT)-Friendly Application Design Guidelines. RFC 3235 (Informational), Leden 2002.
URL http://www.ietf.org/rfc/rfc3235.txt

[18] Srisuresh, P.; Egevang, K.: Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), Leden 2001.
URL http://www.ietf.org/rfc/rfc3022.txt

[19] Srisuresh, P.; Ford, B.; Sivakumar, S.; aj.: NAT Behavioral Requirements for ICMP. RFC 5508 (Best Current Practice), Duben 2009.
URL http://www.ietf.org/rfc/rfc5508.txt

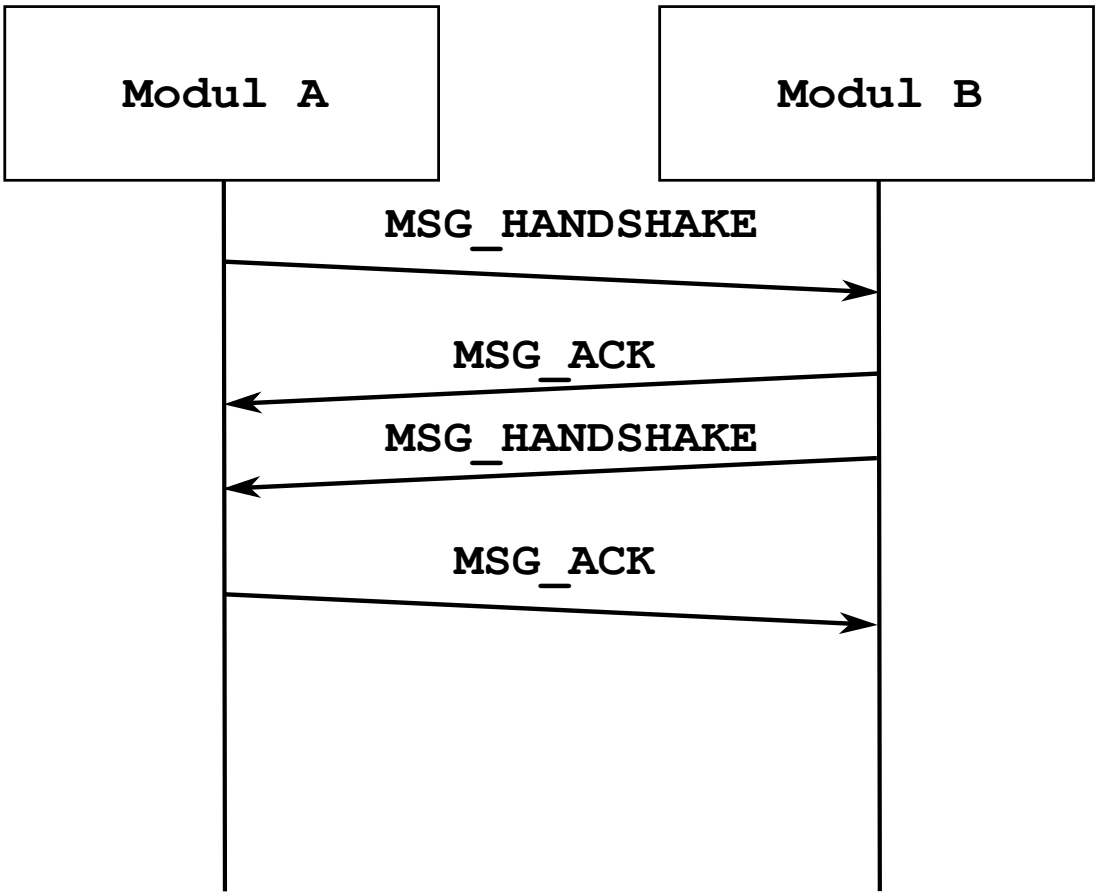
[20] Tom Herbert, W. d. B.: Scaling in the Linux Networking Stack.
URL https://www.kernel.org/doc/Documentation/networking/scaling.txt

Přílohy

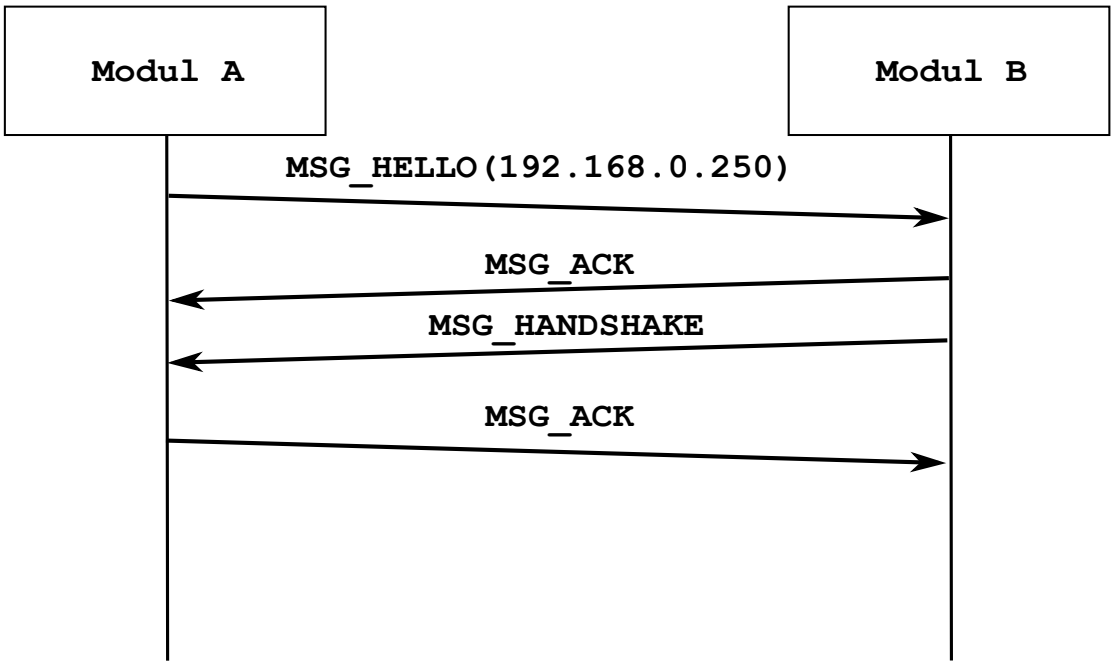
Seznam příloh

Příloha A

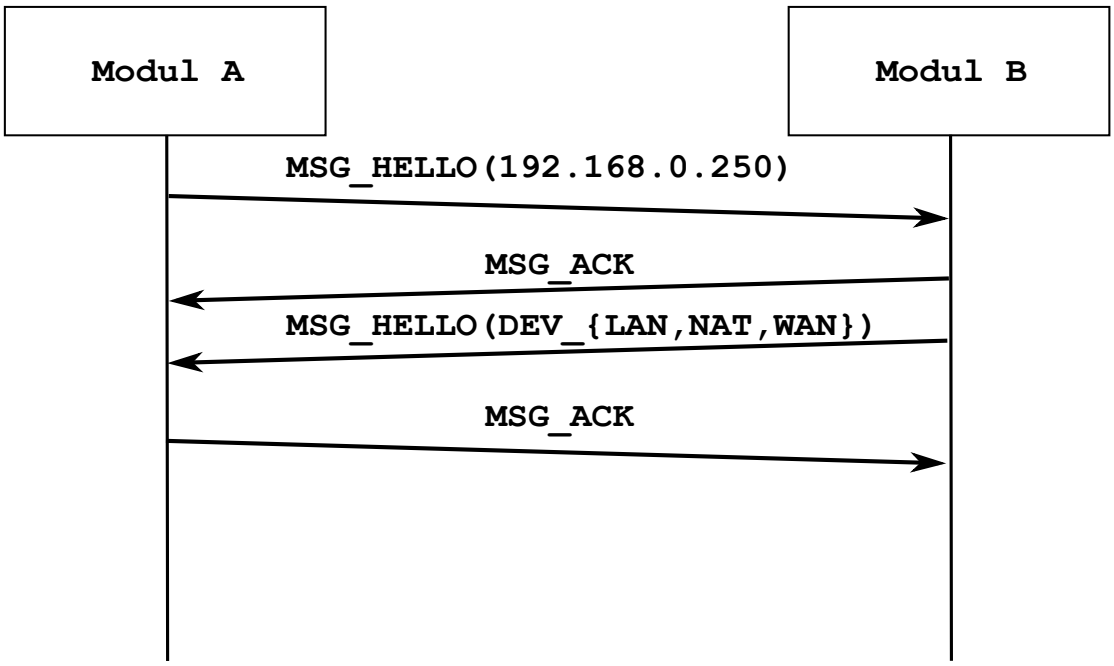
4-way handshake mezi moduly NTF



Obrázek A.1: Základní 4-way handshake mezi moduly



Obrázek A.2: 4-way handshake identifikací klienta



Obrázek A.3: 4-way handshake identifikací klienta a přiřazením role

Příloha B

HW parametry testovacích zařízení

Klientská stanice

- CPU: Intel Xeon E5420@2.50 GHz
- Pamět: 10 GB DDR2@667 MHz
- Operační systém: Debian 8.0 "Jessie", jádro Linux 3.16.0
- Síťové rozhraní: Intel Ethernet Server Adapter X520-2, 2x10 Gb

NAT

- CPU: Intel Xeon E5-2620@2.00 GHz
- Pamět: 32 GB DDR3@1600 MHz
- Operační systém: Debian 8.0 "Jessie", jádro Linux 4.3.3
- Síťové rozhraní: Intel Ethernet Server Adapter X520-2, 2x10 Gb

Stanice serveru

- CPU: Intel Core i7-6700@3.40 GHz (Turbo@4.00 GHz)
- Pamět: 16 GB DDR4@2133 MHz
- Operační systém: Debian 8.0 "Jessie", jádro Linux 4.4.0
- Síťové rozhraní: Intel Ethernet Server Adapter X520-2, 2x10 Gb

Příloha C

Skriptovací API pro popis topologie a testovací sady

C.1 Funkce pro popis topologie

```
-- Funkce pro přidání zařízení do topologie
-- Nastavitelné příznaky:
-- 0x1 : Zařízení nemá spuštěný modul ntfobserver
-- 0x2 : Zařízení bude automaticky konfigurováno
add_device(device_name, device_type, device_conf_ip, flags)

-- Funkce pro přidání testovacího rozhraní
-- Nastavitelné příznaky:
-- 0x1 : Na rozhraní bude zachytáván provoz
-- 0x2 : Rozhraní je primární. Bude tedy použito, pokud není definováno
--       přímé spojení mezi dvěma stanicemi
-- 0x4 : Rozhraní je v simulované veřejné síti.
add_interface(device_name, ifc_name, ifc_mac, ifc_ip, netmask, ifc_flags)

-- Funkce pro vytvoření přímého spojení mezi dvěma stanicemi
link_devices(device_1_name, device_2_name, device_1_ifc, device_2_ifc)

-- Funkce pro nastavení parametrů rozhraní pro emulaci WAN spojení
-- Nastavitelné parametry jsou ztrátovost, duplikace a změna pořadí paketů.
-- Při použití změny pořadí je nutné vyplnit parametry
-- rord_rate, rord_corr a dl
emulate_network(device_name, ifc_name, loss_rate,
                dupl_rate, rord_rate, rord_corr, dl)
```

C.2 Funkce pro popis testovací sady

```
-- Funkce pro vložení testu do testovací sady
set_new_test(procedure_path)

-- Funkce pro změnu parametrů testovací procedury
set_test_parameter(param_key, param_value)

-- Funkce pro spuštění testu
start_test()
```

Příloha D

Skriptovací API pro popis testovacích procedur

```
-- Funkce pro spuštění záchytu paketů na stanici.
-- Záchyt bude spuštěn na rozhraních podle topologie.
-- Parametr filter_string odpovídá použitému backendu. Pro pcap
-- tedy platí stejná syntaxe jako pro tcpdump.
start_packet_capture(device, filter_string)
reload_packet_capture(device)

-- Funkce pro simulaci stažení souboru.
perform_download(flags, downloader, server, protocol, filename)

-- Funkce pro simulaci ICMP echo.
perform_icmp_echo(flags, sender, receiver)
perform_icmp_echo_outside(flags, sender, outside_ip)

-- Funkce pro simulaci spojení na určitém portu.
perform_l4_echo(flags, sender, receiver, port, protocol)
perform_l4_echo_outside(flags, sender, receiver_ip, port, protocol)

-- Funkce pro spuštění uživatelského příkazu.
run_user_command(flags, executor, command)
```

Příloha E

Ukázka konfigurace pravidel pro DPDK NAT

```
{
  "inactive_timeout": 300,
  "lan_ip": "10.0.0.2",
  "wan_ip": "1.2.3.4",
  "wan_gateway_ip": "5.6.7.8",
  "wan_mac_addr_spoof": "00:11:22:33:44:55",
  "default_connections_limit": 1000,
  "1_to_M_IP": [ "192.168.1.1", "192.168.1.2" ],
  "1_to_1_IP_restricted":
  [
    {
      "lan":"10.254.0.1",
      "wan":"192.168.1.253"
    }
  ],
  "1_to_1_IP_unrestricted":
  [
    {
      "lan":"10.254.0.2",
      "wan":"192.168.1.254"
    }
  ],
  "dnat":
  [
    {
      "lan_ip":"10.127.0.1",
      "lan_port":1234,
      "wan_ip":"192.168.1.127",
      "wan_port":4321,
      "protocol": "udp"
    }
  ],
```

```
  "static_routes":
  [
    {"subnet": "10.127.0.0", "mask":"255.255.0.0", "gateway":"10.20.30.40"},
    {"subnet": "10.254.0.0", "mask":16, "gateway":"50.60.70.80"}
  ],
  "per_client_connections_limit":
  [
    {"lan_ip":"10.254.0.2" ,"limit":1000}
  ]
}
```