VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE PODPORY NÁSTROJE NETEM V PROJEKTU LNST

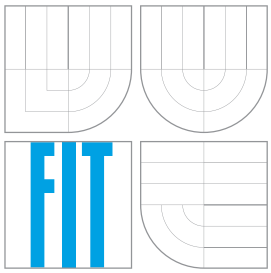
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ PROCHÁZKA

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

IMPLEMENTACE PODPORY NÁSTROJE NETEM V PROJEKTU LNST

IMPLEMENTATION OF NETEM SUPPORT FOR LNST

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ PROCHÁZKA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ONDREJ LICHTNER

BRNO 2015

Abstrakt

Tato práce se věnuje emulacím sítí. Obsahuje uvedení do problematiky emulací a jiných metod experimentování. Součástí je také výpis a rozbor softwarových a hardwarových emulátorů. Zvláštní důraz je kladen na emulátor NetEm, který je použit v praktické části práce. Ta obsahuje základní informace o aplikaci LNST, do které byla implementována podpora emulátoru NetEm. Nakonec pak byla vytvořena sada testů využívající tuto implementaci.

Abstract

The main focus of this thesis is on network emulation. It contains introduction to the field of emulation and different experimentation methods. Part of this work is also devoted to the analysis of software and hardware emulators. Special emphasis is placed on the NetEm emulator, which is used in the practical part of this thesis. Practical part is focused on implementation of NetEm support for application LNST. Finally, test suite which is using this new implementation was created.

Klíčová slova

emulace sítí, emulátory, NetEm, simulace, traffic control, LNST, testování

Keywords

network emulation, emulators, NetEm, simulation, traffic control, LNST, testing

Citace

Jiří Procházka: Implementace podpory nástroje NetEm
v projektu LNST, bakalářská práce, Brno, FIT VUT v Brně, 2015

Implementace podpory nástroje NetEm v projektu LNST

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval sám pod vedením Ing. Ondreje Lichtnera.

.....
Jiří Procházka
20. května 2015

Poděkování

Děkuji vedoucím mé bakalářské práce Ing. Ondreji Lichtnerovi a Ing. Janu Tlukovi, za cenné rady a trpělivost a ochotu mi pomáhat se všemi aspekty mé práce.

Obsah

Kapitola 1

Úvod

Testování je obor, který je lidstvu znám od nepaměti. Už v prehistorických dobách lidé testovali, aniž by si toho byli vědomi. Například při vytváření primitivních nástrojů jako jsou nože, sekery, kladiva, apod., museli otestovat jejich funkčnost před tím, než je začali používat. Tento způsob testování, testování v reálném světě, se tak stal první etapou tohoto oboru. O několik miliónů let později se tento obor posunul dále, když bylo vynalezeno analytické modelování, ve kterém se objekty reálného světa zaměnily za matematické modely. Postupem času se ukázalo, že přesně modelovat složité systémy je obtížné a výsledky z nekvalitního modelu neodpovídají realitě dostatečně. S nástupem počítačů se zavedla nová metoda testování – simulace. Simulaci lze chápat jako experimentování s logickými modely systémů a interpretací výsledků. Tyto modely jsou spuštěny na počítačích, díky kterým je možné vyhodnocovat chování složitějších systémů za kratší čas než tomu bylo u ostatních metod. Emulace je technikou, která spojuje simulace a testování v reálném světě. Klíčovou myšlenkou emulace je reprodukování v reálném čase, v kontrolovaném prostředí, čímž lze pozorovat funkcionalitu sledovaných prvků modelovaného systému při interakci s reálnými systémy. Emulace sítí se zabývá jak síťovými zařízeními, tak i jejich vzájemnou komunikací. Tím, že jsou při emulaci sítě používány reálná zařízení spolu se simulovanými, dosahuje emulace dostatečně věrohodných výsledků. Díky tomu lze prohlásit, že emulace sítí je účinný nástroj pro vyhodnocování testů síťových zařízení, protokolů a aplikací.

Cílem této práce je implementovat emulátor WAN[1] sítí a sadu testů, která bude obsahovat různé scénáře, jenž mohou nastat při přenosu dat ve větších sítích. Emulátor bude implementován jako modul pro aplikaci LNST[2], který se zabývá vytvářením, spouštěním a vyhodnocováním automatizovaných testů pro různé síťové topologie a pro různé typy síťových zařízení. Aplikace *NetEm*[3], která umožňuje ovlivňovat síťový provoz vychazející směrem ven ze síťových rozhraní, bude použita pro simulaci artefaktů, které mohou při přenosu ve WAN sítích nastat. Pomocí těchto dvou aplikací vytvořeny testovací sady tak, aby odpovídaly podmínkám, které v reálných sítích mohou nastat. Na spuštění a vyhodnocení testů navazuje zamyšlení nad využitím této práce v praxi a návrh, jak by se práce dala rozšířit.

Práce je členěna do několika částí. Kapitola 2 je úvodem do emulace sítí, jsou zde popsány základní metodiky používané v minulosti a v současnosti. Kapitola obsahuje přehled existujících emulátorů, softwarových a hardwarových, komerčních a nekomerčních. V kapitole 3 je rozebrána aplikace *NetEm*. Je zde stručně popsána její implementace a příklady

[1] Wide Area Network
[2] Linux Network Stack Test
[3] zkráceno z Network Emulation

spouštění. V kapitole 4 je popis aplikace *LNST*, je zde popsána stručně jeho implementace a možnosti využití v praxi. V kapitole 5 je stručný popis implementace praktické části této práce, tzn. podpory aplikace *NetEm* do aplikace *LNST*. Kapitola 6 je shrnutím procesu vytvářením testovacích sad, jejich zhodnocením a popis užití. Poslední částí je závěr, ve kterém je shrnuto, co tato práce dokázala a kam by se v případě rozšíření mohla ubírat dále.

Kapitola 2

Emulace sítí

V této kapitole je do detailu rozebrána emulace sítí. Na začátku je krátká definice s popisem, co si představit pod pojmem emulace a jaký je rozdíl mezi emulací, simulací a dalšími metodami testování. Dále je zde stručně popsána historie samotné emulace sítí, ze které lze zjistit, jaké vlastnosti jsou předmětem zkoumání a proč. Poté následuje zhodnocení, v čem jsou emulace užitečné a proč je pro tuto práci přínosnější použít emulaci a ne některou z ostatních metod experimentování. A v samotném závěru této kapitoly je rozbor několika emulátorů. Tento rozbor je rozdělen do dvou podkapitol, která rozlišuje mezi softwarovými a hardwarovými emulátory a obě z těchto kapitol jsou ještě rozděleny na komerční emulátory a emulátory, které jsou dostupné zdarma. Při psaní této kapitoly jsem čerpal z knihy Introduction to Network Emulation[15].

2.1 Emulace

Emulace jako taková není pojem, který se vztahuje pouze k oblasti počítačů. Často se používá například pro trénování aktivit, které by byly příliš riskantní, nákladné nebo neetické, pokud by byly prováděny nad reálnými objekty. Asi těžko si lze představit, že by se například v jaderných elektrárnách zkoušelo přímo na běžícím reaktoru. A nebo aby první testovací let pilota byl ve stroji za několik miliónů dolarů, několik kilometrů nad zemí a s vážným rizikem, že nový pilot let nezvládne a havaruje. Právě letecké trenažery jsou asi nejznámějším příkladem emulátoru.

2.1.1 Historie

První experimenty s emulací sítí byly uskutečněny v roce 1980. Na dnešní dobu se jednalo o primitivní způsob testování, přesto mělo obrovský dopad na všechny později prováděné experimenty. Testoval se tehdy relativně nový protokol TCP/IP[1] (ten uvedli Vint Cerf a Bob Kahn v roce 1974). Zjišťovalo se, jak se tento protokol dokáže vypořádat s neideální sítí. Pro experiment byl vymezen pojem „flakeway“. Jednalo se o bránu (gateway), na které bylo možné ovlivňovat parametry toku, který jí procházel. Nastavovat se dalo jaké procento datagramů bude zahozeno, zpožděno, poškozeno, duplikováno či přeuspořádáno. Tyto vlastnosti měly simulovat tzv. degradovanou síť. Ta vzniká kvůli fyzikálním zákonům, které nelze obejít. Například protože přenos signálů trvá nenulovou dobu, vzniká zpoždění přenosu. V sítích, kde data prochází velkým množstvím zařízení, se navíc musí brát zřetel na

[1] Transmission Control Protocol/Internet Protocol

kapacity těchto zařízení. Tím se zhoršuje propustnost celé sítě a případně vznikají i ztráty paketů.

Další experimenty stojící za zmínku proběhly v roce 1995. Byly také zaměřeny na protokol TCP/IP, konkrétně se testovaly jeho dvě nové varianty, TCP Vegas a TCP Reno. Rozdíl oproti experimentům z roku 1980 spočíval především v tom, že při těchto experimentech byla degradace kvality sítě ovlivňována jednotlivými koncovými zařízeními a ne bránou. To bylo umožněno úpravou operačního systému, který na stanicích běžel. Podařilo se tak v lokální síti simulovat degradace, které za normálních podmínek vznikají především ve WAN sítích[5].

Ačkoliv samotná myšlenka emulace sítí vznikla před více než 30-ti lety a emulace se provádí déle než 10 let, stále není přesně definováno, co si pod tímto pojmem představit. Většina výzkumníků se však shodla na tom, že emulace sítí zahrnuje reálné prvky použité v kombinaci se simulovanými komponentami. Je to kompromis mezi simulací a testováním v reálných podmínkách na reálných zařízeních.

2.1.2 Odlišné přístupy k testování

V této podkapitole jsou popsány různé přístupy k experimentování v oblasti sítí a jejich srovnání s emulací. Každý přístup vyžaduje znalosti z různých odvětví, proto je potřeba si u každé sekce nadefinovat pár výrazů, které se v dané oblasti používají.

Analytické modelování

Tento přístup je od zbylých odlišný především v tom, že se jedná o myšlenkový experiment a s ničím se reálně neexperimentuje. Jelikož z něj ostatní metody vycházejí, je vhodné si jej zde pro úplnost uvést.

Definice 1. *Analytické modelování v oblasti počítačových sítí lze chápat jako myšlenkový experiment, který používá pouze analytické rovnice k odhadu chování daného systému, aplikací nebo protokolů.*

Analytické modelování se v sítích nejčastěji vyskytuje ve spojení s frontami, které se používají například u směrovačů a přepínačů. Bohužel z modelu samotného nelze určit, jestli se systém bude chovat podle našich požadavků. Lze jej ale použít pro základ simulace nebo emulace. Hlavním problémem, se kterým se modelování potýká, je že pro matematický popis reality je potřeba realitu abstrahovat. A model nemůže garantovat, že popis chování abstrahovaného systému, bude stejný s popisem chování vzorového.

Výhodou je nízká cena, k získání výsledků stačí pouze počítač, který dokáže řešit numerické modely. Osoba provádějící experimenty má plnou kontrolu nad podmínkami modelovaného systému. Rozsah vlastností, které lze zkoumat je velký. Věrohodnost výsledků je nízká a závisí na kvalitě odvozeného modelu. Jelikož reálné systémy v sobě mají vždy určitou míru variability, kterou nelze promítnout do modelu, vzniká tak model, který není izomorfní s reálným systémem jenž modeluje. Výsledek proto nemusí a zpravidla ani nebývá příliš přesný. Velkou výhodou analytických modelů je jejich relativně snadné použití. Stačí sestavit rovnice, nechat je spočítat a výsledky jsou známy během několika sekund. Problémem může být interpretování výsledků, jelikož vztahy mezi matematickým modelem a realitou může být někdy těžké nadefinovat.

Počítačová simulace sítí

Definice 2. *Počítačová simulace sítí je typ experimentování, používající pouze počítačové modely síťových systémů, aplikací a protokolů.*

Počítačové simulace jsou oblíbené především díky pohodlnému ovládání. Navíc je možné je pouštět na jednom počítači, čímž se redukují náklady, které by jinak bylo potřeba vynaložit na vybudování celé síťové infrastruktury, která se pro každý scénář může lišit. Simulátory existují jak bezplatné tak komerční a to v různých cenových relacích. Používají diskrétní simulaci. Tento typ simulace popisuje fungování systému jako chronologicky uspořádanou sérii událostí. Každá událost má nulovou dobu provedení a mění stav systému. V simulaci se zpravidla používá modelový čas, protože diskrétní čas se nedá navázat na čas reálný.

Simulace vznikají vytvořením modelu, jehož chování co nejvíce odpovídá modelovanému systému a následně vytvořením spustitelného kódu, nejčastěji použitím kompilovaného jazyka. Alternativou může být použití vysokoúrovňových skriptovacích jazyků (např. Python), které jsou interpretovány a spouštěny na počítači bez nutnosti překladu. Modeluje chování všech prvků systému, fyzických (kabely, fronty na přepínačích) i logických (protokoly, aplikace).

Cena počítačových simulací je variabilní. Lze například použít jeden počítač a software, který je zdarma nebo více počítačů a komerční placené aplikace. Díky zavedení logického času lze dobu simulace ovlivňovat. V nezajímavých úsecích experimentu můžeme čas zrychlit, naopak v důležitých okamžicích jej zase zpomalit. Celková doba simulace by se tím však měla podstatně zkrátit. V počítačových simulacích má uživatel kompletní kontrolu nad vlastnostmi prostředí, v němž simulaci provádí. Rozsah vlastností, které můžeme zkoumat je, stejně jako u analytického modelování, velký. Simulátory většinou zaberou nějaký čas, než uživatel plně pochopí zacházení s nimi, avšak po naučení s nástrojem se doba vytváření experimentu výrazně zkracuje. Uživateli stačí pouze napsat scénář a spustit jej. Výsledky bývají snadno interpretovatelné, jelikož se vždy jedná o virtuální reprezentaci reálných zařízení.

Testování v reálném světě

Tato metodika se používá k ověření, že charakteristiky systému, aplikací a protokolů odpovídají příslušnému návrhu.

Definice 3. *Testování v reálném světě je technika experimentování, která používá k experimentům výhradně reálné systémy, aplikace a protokoly.*

Pro většinu experimentů se používají také zařízení, které mají za úkol měřit a analyzovat výsledky experimentu. Tyto zařízení v ostrém provozu nasazeny nebývají. Výsledky těchto experimentů lze většinou brát jako konečné a pokud se v nich nenajde chyba, systém bývá vydán (v případě software) a nebo se zahájí velkovýroba (v případě hardware).

Cena experimentů bývá vysoká z několika důvodů. Je potřeba použít reálná zařízení a tyto zařízení musí být někým nakonfigurována a obvykle pro samotný experiment musí být přítomno více osob, které na ně dohlížejí a spravují je. Většinou uživatel nemá naprostou kontrolu nad jednotlivými prvky, například nelze jednoduše změnit vlastnosti existujícího síťového zařízení, popřípadě na místě provádět větší úpravy protokolů a aplikací. Stejně tak lze pozorovat jen určité aspekty systému, které se odvíjí od konkrétního prostředí. Záleží tak na monitorovacích nástrojích, které jsou pro experiment nasazeny. Věrohodnost výsledků

je na velmi vysoké úrovni, protože test probíhá na reálných zařízeních v reálném prostředí. Neznamená to ale, že tato metoda experimentování odhalí všechny chyby, protože experiment nemusí zahrnovat všechny scénáře, které mohou nastat. Práce s reálnými zařízeními je náročnější než práce se simulacemi a modely a při rozsáhlejších experimentech znatelně narůstá i práce spojená s konfigurací a obhospodařováním zařízení.

2.1.3 Charakteristiky emulace

Emulace je hybridní metodika, která spojuje simulaci a testování v reálném světě. Kvůli tomu je většina vlastností emulace kompromisem mezi vlastnostmi simulace a testování v reálném světě. Cena emulace bývá větší než u simulace, ale je menší než při testování v reálném světě. Vše se odvíjí od poměru fyzických a virtuálních prvků v systému. Doba experimentu většinou odpovídá době, kterou by zabralo testování v reálném světě. V některých případech je možné s časem manipulovat, což je výhoda oproti testování v reálném světě. Kontrolu nad prostředím má uživatel stejnou jako u simulací a větší než při experimentování v reálném světě. Díky možnosti měnit vlastnosti virtuálních prvků se rozšiřuje rozsah zkoumatelných vlastostní systému. Emulace je věrohodnější než analytické modelování a musí být minimálně stejně tak věrohodná jako je simulace. Při emulaci odpadá nutnost budovat celou infrastrukturu z reálných zařízení, díkž čemuž je cena menší. Některá fyzická zařízení přítomna být musí a ty je nutné konfigurovat a spravovat po čas experimentu.

2.2 Emulátory

Emulátorů existují desítky a všechny je popsat by zabralo minimálně jednu samostatnou publikaci. V této kapitole je rozebráno několik emulátorů, které byly vybrány tak, aby z kategorií komerční/neplacené, software/hardware, byl uveden alespoň jeden zástupce.

K emulaci se dá přistupovat různě. Můžeme si sami zvolit poměr mezi prvky simulovanými a reálnými. To ovlivňuje nejen výsledky experimentu, ale také dobu trvání a nebo cenu. Tyto vlastnosti jsou ovlivňovány dalšími faktory, jako je užití distribuovaného/centralizovaného systému spouštění experimentu nebo použití software/hardware emulátorů.

Možností jak klasifikovat emulátory je spousta a pro různé druhy experimentů mohou být vhodné různé typy emulátorů. V této kapitole jsou popsány základní rozdíly mezi emulátory placenými a neplacenými a také rozdíly mezi software a hardware emulátory. V každé sekci jsou uvedeny příklady emulátorů spadajících do příslušné kategorie. U software a hardware emulátorů je přítomna podrobnější analýza. Zvláštní důraz je kladen na software emulátor *NetEm*, který byl použit při implementaci v praktické části této bakalářské práce. Pro něj je vyhrazena samostatná kapitola.

2.2.1 Cenová dostupnost emulátorů

Ačkoliv cena není hlavním kritériem, přesto hraje podstatnou roli při výběru vhodného emulátoru. Emulátory tak můžeme dělit následovně.

Neplacené emulátory

Tyto emulátory jsou dostupné ke stažení zdarma. Často bývá jejich součástí kompletní zdrojový kód, uživatelé si tak mohou přidávat svoje vylepšení nebo aplikovat svoje opravy chyb nebo si program měnit podle svých specifických požadavků. Neplacené emulátory

bývají zpravidla jednoduché na instalaci a používání. Nevýhodou bývá minimální technická podpora ze strany výrobce, uživatelé se tak musí spoléhat na komunitu uživatelů, kteří daný produkt používají.

Emulátory sloužící pro výzkum

Do této kategorie spadají emulátory, které jsou dostupné zdarma pouze pro výzkumné účely. Bývají náročné na instalaci, konfiguraci i obsluhu. Pro využití v komerčním prostředí bývá vyžadována licence, která je placená. Technická podpora ze strany výrobce většinou existuje, ale její kvalita je limitována nízkým počtem pracovníků, kteří se jí věnují.

Komerční emulátory

Sem se řadí všechny emulátory, které se používají v komerční sféře. Zpravidla bývají snadno instalovatelné i obsluhovatelné a k dispozici je kvalitní dokumentace a rozšířená technická podpora. Zdrojové kódy emulátorů zveřejněny nebývají, v některých případech je však možné je obdržet po domluvě s výrobcem.

2.2.2 Software emulátory

Definice 4. *Software emulátor je takový emulátor, který existuje pouze v elektronické podobě a vyžaduje hardware na kterém může běžet ke spuštění. Typicky se jedná o počítač nebo ekvivalentní platformu.*

Funkcionalitu software emulátorů implementuje počítačový program. Někdy výrobci emulátorů prodávají také upravený laptop, s předinstalovaným emulátorem, jako tzv. „out of box“ řešení. Většinou je však na uživateli, aby si potřebný hardware obstaral sám. Tento druh emulátorů je používán nejčastěji, obzvláště ve výzkumu. Důvodem je jejich cena, mnoho existujících řešení je možné používat zdarma. Výkon se odvíjí od hardwaru, na kterém běží, to ale zpravidla nebývá příliš limitující. Typické pro tyto emulátory je snadná instalace, některé (například Dummynet a NetEm) jsou součástí některých operačních systémů. Neplacené emulátory ovšem nenabízejí tolik možností konfigurace jako ty komerční.

Dummynet

Dummynet[6] byl vyvinut v druhé polovině 90. let na univerzitě v Pise za účelem testování protokolů. Dokáže nastavovat vlastnosti front, propustnost linky a zpoždění a ztrátovost paketů. Původní verze používala algoritmus WF^2Q+[7], v nových verzích je jich na výběr více. Dá se použít na koncových stanicích pro ovládání toků směřujících do a z počítače, ale i na strojích, sloužících jako směrovače nebo bridge, kde se mění data procházející přes dané síťové zařízení. Původní implementace běžela pouze na FreeBSD, postupem času byl Dummynet naprogramován i pro operační systémy OS X, Windows a Linux. Dummynet je možné použít při emulacích centralizovaných i distribuovaných. V případě centralizované emulace je potřeba Dummynet pouštět na stroji, který má minimálně dvě síťová rozhraní a chová se podobně jak směrovač - na jedno rozhraní datagramy přicházejí, stroj nad nimi provede plánované operace a druhým rozhraním odcházejí. Při distribuované emulaci se Dummynet pouští na všech stanicích a provoz je ovlivňován přímo na nich. Dummynet je zdarma a přestože nenabízí mnoho možností nastavení, na jednodušší experimenty stačí. V nových verzích navíc byla přidána podpora pro bezdrátové sítě[8].

Shunra VE Desktop

Shunra je společnost, která poskytuje softwarové a hardwarové emulátory. Jejich řešení umožňují emulaci celého síťového prostředí a jsou určena spíše pro firmy než na výzkum. U software emulátorů Shunra nabízí hned několik variant: Shunra VE Cloud, Shunra VE Desktop Standard a Shunra VE Desktop Professional. Desktop Standard verze je určena pro operační systém Windows a umožňuje měnit základní parametry přenosu (ztrátovost, zpoždění, šířku pásma) pomocí jednoduchého grafického rozhraní. Používá se k rychlé základní transformaci LAN sítě na WAN síť a umožňuje vývojářům otestovat, jak se bude jejich aplikace chovat v ostrém provozu. Professional verze poskytuje větší možnosti nastavení a také věrohodnější emulaci, která nabízí přesnější výsledky. Ve verzi Professional lze emulovat i složitější scénaře a nabízí například automatickou analýzu experimentu, integrace s jinými nástroji od firmy Shunra a možnost rozšiřování pomocí otevřeného API[2]. Hlavním cílem těchto emulátorů je především poskytnout vývojářům prostředí podobné WAN sítím pro otestování jejich aplikací. Nejsou příliš vhodné pro výzkum, protože neumožňují detailní nastavení testovacího prostředí.

2.2.3 Hardware emulátory

Definice 5. *Hardware emulátor je síťový emulátor ve formě hardware zařízení, který sám o sobě poskytuje kompletní funkcionalitu emulování.*

Ačkoliv i software emulátor může být dodáván jako hardware s předinstalovaným emulovacím softwarem, hlavní rozdíl oproti hardware emulátoru je ten, že hardware emulátor slouží pouze k emulování a operační systém, na kterém běží bývá pro tento úkol maximálně optimalizován. Používají se také emulátory stavěné na FPGA, které nabízejí vyšší rychlost a přesnost výsledků, ale nevýhodou bývá jejich vyšší cena. Hardware emulátory mívají více možností konfigurace testovacího prostředí než software emulátory, obzvláště než ty neplacené. Asi největší výhodou, kterou hardwarová řešení nabízejí na rozdíl od softwarových, je možnost pracovat s linkami s vyššími přenosovými rychlostmi. Mnoho firem, které nabízí komerční software emulátory, nabízí zároveň také hardware emulátory. Tyto emulátory většinou sdílejí funkcionalitu, ale hardware emulátory najdou uplatnění spíše v sítích s vyššími přenosovými rychlostmi a v experimentech, kde záleží na přesnosti výsledků.

Shunra VE Appliance

Shunra VE Appliance je vlajkovou lodí emulátorů této firmy. Umožňuje vytvoření celé virtuální sítě ve které se pak provádějí experimenty. Produkty firmy Shunra kromě testovacího prostředí poskytují i nástroje pro analýzu výsledků. Součástí je webové grafické rozhraní pro správu a konfiguraci experimentů. Tento emulátor poskytuje rozhraní s různými přenosovými rychlostmi, od 10 Mbps až po 10 Gbps, s celkovou přepínací kapacitou 24 Gbps. Umožňuje nastavovat velké množství parametrů, například: zpoždění, omezení šířky pásma, ztrátovost paketů, přeuspořádávání paketů, výpadky linky, apod. Lze emulovat různé topologie, jako klient-server, full-mesh sítě, topologie podobné Internetu či topologie používané v komerčním prostředí. Zvládá také různé technologie, které se ve WAN sítích používají: ethernet, frame relay, bezdrátové sítě, satelitní sítě, IPv4 i IPv6. Právě díky širkému spektru vlastností, které se v tomto emulátoru dají nastavovat je vidět, proč jsou lepší než jejich software alternativy.

[2] Application Programming Interface

V roce 2014 byla firma Shunra odkoupena firmou HP[10] a produkty přejmenovány.

PacketStorm 4XG

PacketStorm Communications[3] je další velká firma, která se zabývá hardwarovými a softwarovými emulátory. PacketStorm 4XG je hardware emulátor, jehož architektura umožňuje propustnost až 40 Gbps a zpracování 64 miliónů paketů za vteřinu. Umožňuje emulovat deformace paketů a také jevů vyskytujících se v sítích. Naměřená data si ukládá, zpracovává a uživateli umožňuje výsledky prohlížet pomocí webového rozhrání. 4XG podporuje mimo jiné: zpoždění paketů, ztrátovost paketů (náhodná i dávková), duplikaci paketů, přeuspořádávání paketů, fragmentování paketů, úprava paketů v reálném čase a jitter. Jitter je kolísání velikosti zpoždění paketů při průchodu sítí[4]. Umí také provádět filtrování paketů podle údajů vyskytujích se v hlavičce paketu. Ovládá se prostřednictvím webového rozhrání nebo vzdáleně pomocí sady TCL příkazů. Vysoký výkon 4XG se dá využít především v těchto scénářích: vojenská bezpečnost, síťová bezpečnost, video aplikace, síťová úložiště, atd.

Kapitola 3

NetEm

Jak zde bylo řečeno, při implementaci emulátoru pro praktickou část této práce byla použita aplikace *NetEm*, proto je jí věnována celá kapitola a samotná aplikace bude rozebrána detailněji než ostatní emulátory.

Název *NetEm* vychází ze spojení slov Network Emulator. Jedná se o jednoduchý softwarový emulátor pro operační systém Linux. V některých distribucích (Fedora, OpenSUSE, Gentoo, Debian, Mandriva, Ubuntu) je *NetEm* součástí kernelu od verze 2.6. Ovládá se pomocí programu příkazové řádky *tc*. Aktuální verze umožňuje emulovat zpoždění, ztrátovost, duplikaci a přeuspořádání paketů. Při tvorbě této kapitoly bylo čerpáno z těchto zdrojů[15, 12, 9].

3.1 Popis nástroje

NetEm je součástí jedné z komponent programu *traffic controller*. Síťový provoz je možné ovlivňovat těmito metodami.

Shaping

Mění rychlost vysílání síťového toku. Neznamená to, že by propustnost pouze snižoval, ale používá se například ke zjemnění nárazů v síťovém provozu. To má za následek snížení pravděpodobnosti zahlcení síťových zařízení. Shaping se provádí na výstupních datech.

Scheduling

Scheduling se používá k plánování vysílání paketů, čímž zajišťuje interaktivitu provozu, který to potřebuje například z důvodu náhlého vyslání většího objemu dat s garantovaným přenosovým pásmem. Změna uspořádání paketů se také nazývá prioritizování a odehrává se pouze u výstupních dat.

Policing

Používá se k nastavování vlastností příchozích dat v podobném smyslu, jako se shaping používá u výstupních dat.

Dropping

Dropping znamná zahazování datagramů, které přesahují povolenou šířku pásma. Zahazování se může odehrávat na vstupu i výstupu.

3.1.1 Zpracování provozu

Při zpracovávání dat se používají tři objekty - qdiscy, třídy a filtry. Qdisc je zkrácený výraz queueing discipline a je elementárním prvkem při zpracování provozu. Vždy když chce linuxový kernel odeslat paket na síťové rozhraní, je nejdříve zařazen do příslušné fronty. Nejjednodušší typ fronty je FIFO[1], ten však neposkytuje žádné možnosti manipulace s daty. *NetEm* proto změnil způsob jakým fronty fungují a tím umožňuje ovlivňovat některé parametry přenosu, jako například zpoždění paketu či jeho zahození. Qdiscy mohou obsahovat třídy, které mohou obsahovat vnořené qdisky. Při umisťování paketů do fronty, se volí některá z vnitřních front, čímž vzniká prioritizování určitého toku. K tomu, aby se určilo, do které fronty bude paket zařazen slouží filtry.

3.2 Konfigurace

NetEm se konfiguruje pomocí programu příkazové řádky *tc*. Za něj se přidávájí parametry *qdisc* a *netem*. V této podkapitole jsou rozebrány konfigurace jednotlivých artefaktů.

3.2.1 Zpožďování paketů

Umožňuje nastavovat zpoždění příchodu paketů na síťové rozhraní. Na výběr máme několik způsobů, jak se zpožděním manipulovat:

- Fixní hodnota zpoždění, která se aplikuje na všechny pakety.
- Hodnoty zpoždění pro každý paket jsou náhodně generovány z určitého intervalu. Tohle chování je pro WAN sítě typičtější, protože velikost zpoždění paketu se náhodně mění.
- Náhodné generátory mohou používat různé matematické funkce. K dispozici jsou tyto typy distribucí: normální, Paretovo a Paretovo normální.
- Korelované hodnoty zpoždění, které se snaží napodobovat chování zahlcení. Hodnota zpoždění je náhodně generována, avšak s ohledem na předchozí hodnotu.

Ukázka spouštění

Následující příkaz zpozdí všechny příchozí packety z rozhraní eth0 o 100 ms.

```
# tc qdisc add dev eth0 root netem delay 100ms
```

Při spuštění s těmito parametry, bude každý packet zpožděn o hodnotu 100 ms, +/- 10 ms.

```
# tc qdisc change dev eth0 root netem delay 100ms 10ms
```

Zde je aplikována navíc 25% korelace.

```
# tc qdisc change dev eth0 root netem delay 100ms 10ms 25%
```

[1] first in, first out

3.2.2 Ztrátovost paketů

Tady dochází k zahození určitého procenta paketů. Například pokud zvolíme ztrátovost 1 %, průměrně jeden paket ze sta bude zahozen. I zde je možné konfigurovat korelaci, což nám umožňuje věrohodněji simulovat nárazy ztráty dat v určitém okamžiku.

Ukázka spouštění

0.1 % datagramů bude ztraceno.

```
# tc qdisc change dev eth0 root netem loss 0.1%
```

3.2.3 Duplikace paketů

Nastavuje jak velké procento paketů bude zduplikováno. Syntaxe je zde stejná jako u ztrátovosti. Používá se k nasimulování situace, kdy na cílové rozhraní přijde originální paket dvakrát. To může nastat například u protokolu TCP, když je doba přenosu příliš velká. Protokol jej bere jako ztracený a je opětovně zaslán. U mnoha aplikací a protokolů tato situace musí být správně ošetřena, aby nedocházelo k nedefinovanému chování.

Ukázka spouštění

1 % paketů bude zduplikováno.

```
# tc qdisc change dev eth0 root netem duplicate 1%
```

3.2.4 Poškozování paketů

Používá se k emulaci poškozování paketu, které může vzniknout například šumem ve vodiči. Tím, že se změní i jeden bit, se paket stává nevalidním a bývá zahozen. Šance na vznik takové chyby je sice minimální u drátových sítí, u bezdrátových však tento jev lze pozorovat celkem často v důsledku rušení signálu. Tato vlastnost se tedy používá především u emulací bezdrátových sítí a k otestování, zda se testovaný objekt dokáže vypořádat s chybným paketem správně.

Ukázka spouštění

0.1 % datagramů bude poškozeno.

```
# tc qdisc change dev eth0 root netem corrupt 0.1%
```

3.2.5 Přeuspořádání paketů

Ve WAN sítích často nastává situace, že pakety k cíli cestují různými cestami a tak se může lišit i pořadí v jakém do cíle doputují. Tomuto jevu se říká přeuspořádání paketů (anglicky packet reordering) a protokoly a aplikace se s ním musí umět vypořádat. *NetEm* umožňuje specifikovat, jak velké množství paketů bude přeuspořádáno a to buď procentuelně nebo fixní hodnotou.

Ukázka spouštění

Každý pátý paket bude odeslán okamžitě, ostatní se zpožděním 10 ms. Tento parametr musí být zadán v kombinaci s parametrem *delay*.

```
# tc qdisc change dev eth0 root netem gap 5 delay 10ms
```

3.2.6 Kontrola propustnosti

NetEm umožňuje limitovat propustnost na zařízení. K tomuto omezování používá Token Bucket filtr, protože jiné objekty neumožňují efektivně měnit šířku přenosového pásma.

3.3 Zhodnocení

NetEm je software emulátor, který je zdarma a nabízí dostatečnou škálu parametrů, které lze nastavovat. V základu lze sice ovlivňovat pouze odchozí provoz, toto omezení lze však obejít použitím tzv. IFB[2].

[2]Intermediate Functional Block

Kapitola 4

LNST

LNST, celým názvem Linux Network Stack Test, je aplikace pro automatizované spouštění testů zaměřujících se na linuxovou síťovou implementaci. Motivací pro vznik této aplikace byla snaha rozšířit pokrytí testy a zároveň mít nástroj, který by umožňoval jednoduše spouštět a vyhodnocovat sady regresních testů a testů výkonnosti, bez nutnosti ručně konfigurovat celé testovací prostředí. Tato aplikace je napsaná v jazyce Python verze 2.7. Jedná se o open source projekt, který je zdarma dostupný ke stažení na adrese `www.lnst-project.org`. Při tvorbě této kapitoly bylo čerpáno z oficiálních a wiki stránek aplikace *LNST*[11, 14]. Nástroj vznikl v roce 2011 pod názvem NetTest. Později byl přejmenován na Linux Network Stack Test, což lépe odráží jeho využití. Je vyvíjen firmou Red Hat v rámci týmu Kernel QE.

Nástroj je vhodný pro vývojáře a testery, kteří pracují v oblasti síťových technologií. V současné době obsahuje sadu asi 25 regresních testů, které pokrývají základní oblasti síťové implementace, jako jsou VLANy, bonding a teaming (active backup[1] a round robin[2]), bridge a jejich vzájemné kombinace, které se v reálných aplikacích mohou objevovat. Kromě detekce regrese se testy mohou použít pro měření výkonu, testy totiž obsahují měření propustnosti pomocí programu *netperf* na protokolech TCP a UDP[3][2]. V rámci základní instalace jsou k dispozici ukázkové testy z wiki stránek projektu a to multicast, smoke a team testy.

Cílem této práce bylo implementovat podporu aplikace *NetEm* do této aplikace, což znamená umožnit nastavovat na rozhraních vlastnosti jako je ztrátovost, zpoždění, duplikace a přeuspořádání paketů a také vytvořit sadu testů, které budou pokrývat různé scénáře, jenž se vyskytují převážně ve WAN sítích a při nestandardních situacích.

4.1 Popis nástroje

V této podkapitole je stručný popis aplikace *LNST*. Jsou zde popsány XML recepty, které jsou používány ke konfiguraci požadované testovací topologie sítě, Python tasky, které umožňují detailněji specifikovat, co se testuje. Následuje stručný popis dvou hlavních komponent, kterými je Controller a Slave. Pro podrobnější informace o aplikaci je uživateli doporučeno nahlédnout do wiki stránek projektu[14].

[1] aktivní je pouze jedno rozhraní, ostatní se zaktivují v případě poruchy hlavního

[2] data jsou na zařízení rozdělována rovnoměrně

[3] User Datagram Protocol

4.1.1 Recepty a tasky

Pro práci s *LNST* je potřeba nadefinovat si dva pojmy, které s *LNST* úzce souvisí: recepty a Python tasky. Recepty jsou psané v jazyce XML a mají dva hlavní cíle:

1. Popsat topologii sítě, ve které se test odehrává. Popisují se jednotlivé stroje a všechna jejich rozhraní, která jsou v rámci testu konfigurována. Na rozhraních se mohou konfigurovat IP adresy, u rozhraní typu team či bond se popisují rozhraní, která jsou agregována a mód, pod kterým daný team/bond běží. Na VLAN rozhraních se konfiguruje VLAN tag.

2. Identifikovat testy, které budou spuštěny. Testovat se může buď v rámci integrovaných či vlastních testovacích modulů nebo pomocí Python tasků. Také se přes ně dají spouštět příkazy a definovat specifickou konfiguraci.

Schéma pro XML recepty je psané jazykem RELAX NG[13] a je obsaženo v souboru recipe-schema.rng. Toto schéma definuje gramatiku, jakými jsou recepty psány a používá se také pro validaci. Pro bližší pochopení, jak psát recepty se uživatelům doporučuje nahlédnout do wiki stránek projektu. Ty jsou k nahlédnutí na adrese www.github.com/jpirko/lnst/.

Pokud uživateli nestačí základní testovací moduly a konfigurace rozhraní pomocí XML receptu, může využít TaskAPI a napsat si přenositelný test v Pythonu. Výhoda Python tasku je především ve větší kontrole nad testem, možnost provádění dodatečné konfigurace rozhraní a to i v průběhu testu a také možnost použití cyklů, podmínek a ostatních řídících struktur tohoto jazyka. Python je také na první pohled přehlednější a srozumitelnější než jazyk XML.

4.1.2 Spouštění testů

Pro spouštění testů je zapotřebí mít připravené síťové prostředí, které bude *LNST* v rámci testu konfigurovat, na počítačích mít nainstalované a spuštěné *LNST* a knihovny, které jsou uvedené v instalačním manuálu. *LNST* je distribuovaná aplikace, kde procesy mohou být spuštěny ve dvou různých rolích, Controller a Slave. Controller běží v rámci jednoho testu napříč všemi počítači pouze jeden, kdežto Slave musí být spuštěn na všech počítačích, které jsou součástí testu. Další podmínkou je, že Controller musí mít se všemi Slave-y funkční síťové spojení, které je využito pro vzájemnou komunikaci přes protokol RPC a které nesmí být součástí testovacího prostředí. Na stroji, na kterém je spuštěn Controller musí být navíc přítomny XML soubory popisující jednotlivé počítače na kterých běží Slave-ové. Tyto soubory obsahují informace jako je doménové jméno, údaj jestli je stroj fyzický nebo virtuální, seznam rozhraní dostupných pro testování, spolu s jejich MAC adresami, podsíť do které spadají a volitelně typ ovladače, který používají. Navíc je zde možné uvést dodatečné parametry, na které lze v receptu s testem odkazovat. Tyhle informace se používají při parsování topologie testovací sítě z receptu, když Controller zjišťuje, jestli je možné danou topologii sestavit ze strojů v jeho tzv. machine poolu[4]. Aplikace podporuje použití virtuálních strojů běžících pod virtualizačním API libvirt[1]. U virtuálních strojů navíc není nutné ručně vytvářet a spravovat rozhraní, *LNST* je samo dokáže dynamicky vytvářet, měnit a mazat podle potřeb daného testu. Virtuální a fyzické stroje ale není možné v rámci jednoho testu kombinovat.

[4] v *LNST* to znamená množinu strojů, které jsou pro testování dostupné

4.1.3 Controller

Controller řídí běh testů. Při spuštění nejdříve zpracuje argumenty z příkazové řádky. Uživatel může pomocí argumentů ovlivnit například výpis debugovacích zpráv, definovaní vlastních aliasů, vytvoření HTML souboru s výsledky běhu, aj.[5] Může si také navolit, zda-li chce testy spustit nebo jen nastavit topologii podle receptu, či jen zkontrolovat, jestli je možné daný test spustit s dostupným machine poolem. Při spuštění je machine pool zkontrolován, na každý stroj je vyslána zpráva přes RPC protokol. Pokud na ni stroj vzápětí odpoví, je to indikátor nejen toho, že je daný stroj dostupný po síti, ale také že na něm běží neobsazený Slave, tzn. je použitelný na testování. Poté se validuje XML recept. Po validaci probíhá kontrola matche, tzn. zda-li je možné přiřadit stroje popsané v topologii v receptu k zařízením, která jsou dostupná v machine poolu. Pokud kontrola proběhne v pořádku, Controller odešle jednotlivým Slave-ům údaje o nastavení jednotlivých rozhraní. Pokud se na Slave-ch podaří všechna rozhraní nakonfigurovat, končí fáze sestavování topologie a na řadu přichází samotný testovací běh. *LNST* momentálně podporuje dva způsoby jak testy zadávat:

1. V rámci XML receptu mohou být uvedeny vytvořené testovací moduly[6] a pomocí XML atributů se nastavují pouze věci specifické pro daný modul, tj. na jakých IP adresách se testuje, jaké jsou hodnoty, pro které je výsledek chápán jako úspěšný a dodatečné parametry k programům, které v rámci testovacího modulu běží.

2. Uživatel si může vytvořit Python task, kde může pomocí TaskAPI[7] detailněji konfigurovat zařízení[8], spouštět svoje vlastní testovací moduly a nebo kombinovat integrované testovací moduly se svými.

Po doběhnutí testu jsou údaje s výsledky odeslány na Controller, který je zobrazuje na standardní výstup. Po doběhnutí všech testů navíc Controller zobrazí souhrnný výpis všech testů a v případě spuštění Controlleru-u s přepínačem `--html` vygeneruje HTML soubor se výsledky testů.

4.1.4 Slave

Slave běží na stroji buď v režimu démona[9] nebo v popředí. Pokud běží v popředí, umožňuje vypisování informací v průběhu testu na standardní výstup, v opačném případě jsou vytvářeny logovací soubory. Slave čeká na spojení od Controller-u na portu specifikovaném v konfiguračním souboru[10]. Po navázání spojení s Controller-em dostane instrukce o konfiguraci jím spravovaného počítače. Pokud při konfiguraci nastane chyba, všechny provedené změny se vrátí do původního stavu a Controller-u je zaslána zpráva s informací o neúspěšné konfiguraci. V opačném případě Slave čeká na další zprávy od Controller-u, ve kterých jsou jednotlivé příkazy, které má Slave spouštět. Výsledky posílá zpět Controller-u a to včetně návratového kódu. Po doběhnutí všech testů dostává Slave pokyn k dekonfiguraci, po které začíná opět čekat na další spojení s Controller-em signalizující nové spuštění testů.

[5] viz nápověda aplikace
[6] např. IcmpPing, Icmp6Ping, Netperf, Iperf, aj.
[7] popis tohoto API je možno nalézt na wiki stránkách projektu
[8] například nastavování MTU na rozhraních momentálně není možné řešit v rámci receptu, v Python tasku to nastavit lze
[9] proces běžící na pozadí
[10] výchozí port je 9999

Kapitola 5

Návrh a implementace

Návrh zahrnoval jakým způsobem se budou *NetEm* parametry na rozhraní aplikovat a vymyšlení formátu, jakým budou zadávány do XML receptů. Samotná implementace pak zahrnovala parsování údajů z XML receptu, jejich odesílání na Slave-y, kompilování *tc* příkazu, jeho spuštění při konfiguraci a také spuštění příslušného *tc* příkazu při dekonfiguraci. Implementace je tak rozdělena do patřičných modulů. Konkrétní úpravy a nové funkce, které byly v rámci praktické části této práce provedeny, lze vypozorovat z git repozitáře, který je součástí CD přikládaného k této práci. Tato implementace bude také aplikována do hlavní větve repozitáře aplikace *LNST*.

5.1 Integrace do XML receptů

Prvním krokem byl návrh integrace do XML receptů s testem. Možností provedení bylo několik:

- Nastavovat *NetEm* parametry v rámci celého stroje - tzn. všechny rozhraní daného stroje budou mít nastavené příslušné *NetEm* parametry.
- Nastavovat je v rámci podsítě, která je identifikována pomocí labelu[1] - parametry by se nastavovaly na všech zařízeních v příslušné síti.
- Nastavovat je na každém rozhraní individuálně.

Po diskuzi se zbytkem vývojářského týmu byla nakonec zvolena třetí možnost, jelikož umožňuje největší kontrolu nad testovacím prostředím. Dalším krokem byl návrh zápisu do receptu. Vybrán byl nakonec tento způsob zápisu:

```
<eth id="testiface" label="testnet">
    <netem>
        <delay>
            <options>
                <option name="time" value="250ms" />
                <option name="jitter" value="25ms" />
                <option name="correlation" value="50%" />
            </options>
        </delay>
```

[1] v tomto kontextu se jedná o identifikaci lokální sítě

```
        <loss>
            <options>
                <option name="percent" value="25%" />
            </options>
        </loss>
    </netem>
</eth>
```

Element eth je rodičem elementu netem a ten je rodičem elementům jednotlivých parametrů, ve kterých se v elementech options nastavují konkrétní hodnoty specifické pro daný *NetEm* parametr. Následovalo rozšíření souboru se schématem popisujícím XML recepty o celou strukturu elementů pro *NetEm* parametry. Implementace je k nahlédnutní v příloze B. Při implementaci jsem vycházel z hotových definicí, které se v souboru nacházejí.

5.2 Controller

Dalším krokem byla implementace extrakce dat z XML receptu, kterou provádí Controller. Byla použita hotová třída a metody pro parsování z XML souboru, navíc byla implementována kontrola validity extrahovaných dat. Vytvořena pro to byla metoda `_validate_netem()`, která je součástí třídy `RecipeParser`. Metoda má za úkol zkontrolovat, které volby pro jednotlivé parametry byly zadány a porovnat je s gramatikou uvedenou v manuálových stránkách nástroje *NetEm*. V případě, že volby *NetEm* gramatice neodpovídají, metoda vyvolá výjimku `RecipeError`, která uživateli zobrazí, který řádek v XML receptu je chybný a proč (příklady chyb: chybí povinná volba, chybně zadaný název distribuce rozložení). Při úspěšné kontrole validity se naparsované údaje přidají do slovníku, který obsahuje konfigurační údaje pro konkrétní rozhraní. Tento slovník s konfigurací se posílá na příslušné Slave-y a to pomocí metody `_rpc_call()`.

5.3 Slave

Slave obdrží pro každé své rozhraní uvedené v receptu zprávu s konfiguračními údaji, které se na rozhraní pokusí naaplikovat. V rámci implementace byla konfigurace rozšířena o kontrolu přítomnosti slovníku s *NetEm* parametry. Dále nastává parsování *NetEm* příkazu, pomocí utility *tc*. Parsování probíhá následovně: kontroluje se přítomnost pole s parametry pro každý možný parametr (packet delay, packet loss, packet corruption, packet duplication, packet reordering), v případě existence se volá samostatná metoda pro parsování. Ta se pomocí nově vytvořené metody `get_netem_option()` pokusí získat hodnotu každé možné volby (podle gramatiky z manuálových stránek) a po získání hodnot vrací řetězec příkazu *tc* pro daný parametr. Výhodou aplikace *tc* je, že všechny *NetEm* parametry je možné vložit do jednoho příkazu a to při aplikaci i odstraňování na/z rozhraní. Po parsování všech parametrů tak máme k dispozici kompletní *tc* příkaz pro nastavení příslušných parametrů.

5.4 Konfigurace a dekonfigurace

Samotné spouštění zkompilovaného *tc* příkazu se provádí v rámci metody `configure`, za pomoci metody `exec_cmd()`. Výhoda použití této metody tkví v tom, že pokud spouštěný

příkaz skončí s nenulovou návratovou hodnotou, automaticky vyvolá výjimku, která obsahuje obsah standardního chybového výstupu. Díky tomuto chování jsou tak automaticky ošetřeny chyby, které by způsobovaly neúspěšné spuštěné aplikace *tc*. Pokud je příkaz spuštěn úspěšně, tzn. hodnota návratové proměnné je rovna nule, celý zkompilovaný *tc* příkaz se uloží do slovníku s konfiguračními údaji pro daný interface. Důvodem je snadnější odstraňování těchto *NetEm* parametrů z rozhraní po ukončení testů. *LNST* totiž po ukončení zpracování XML receptu provádí dekonfiguraci všech zařízení, čímž je vrátí do stavu, ve kterém byly před spuštěním *LNST*. Výjimku tvoří některé příkazy, u kterých je možné nastavit perzistentnost, tzn. aby při závěrečné dekonfiguraci nebyly brány v potaz a zůstaly nastaveny i po ukončení *LNST*.

Dekonfigurace probíhá v metodě `deconfigure()` a to tak, že se nejdříve načte kompletní zkompilovaný *tc* příkaz a přepínač *add* se v něm nahradí přepínačem *del* pomocí metody třídy string `replace()`. Upravený příkaz je předán metodě `exec_cmd()`, která jej vykoná a*LNST* dále pokračuje s dekonfigurací ostatních voleb podle konfigurace ze slovníku. Jelikož *LNST*, z důvodu zachování zpětné kompatibility se staršími systémy, umožňuje konfiguraci rozhraní pomocí programu *ip* i pomocí aplikace *NetworkManager*, v implementaci to muselo být zohledněno a parsování, konfigurace a dekonfigurace jsou tak obsaženy v obou příslušných modulech, `NetConfigDevice` (pro konfiguraci pomocí *ip*) i `NmConfigDevice` (pro konfigurací pomocí *NetworkManager-u*).

Kapitola 6

Testovací sady

Součástí této práce bylo vytvoření sady testů simulující chování sítě Internet, včetně možných nestandardních situací. Právě na tyto nestandardní situace byl kladen důraz. Důvodem je snaha mít nástroj, který by pomohl odhalit nedostatky a chyby v situacích, které nejsou pro síťový provoz běžné a proto při testování v běžném prostředí nemusí být pokryty. V této kapitole je popsána struktura Python tasku, který je společný pro všechny testy a receptů, které se navzájem liší v nastavovaných *NetEm* parametrech, topologie sítě a spouštěný Python task jsou stejné napříč všemi testy.

6.1 Python task a recepty

Python task je společný pro všechny testy vytvořené v rámci této práce. Soubor s Python taskem se jmenuje `netem_test.py` a je umístěn ve složce `recipes/netem/`. Testy jsou strukturizovány následovně: V každém testu jsou dva stroje, každý ze strojů obsahuje jedno ethenetové rozhraní. Parametry programu *NetEm* se konfigurují na ethernetové zařízení vždy pouze jednoho stroje. Rozhraní mají nakonfigurovanou IP adresu z rozsahu 192.168.0.0/24. Součástí testů je test ICMP paketů, kdy pomocí programu *ping* je vysláno 1000 ping paketů na druhý stroj a to v intervalu 0.1s. Poté je na jednom stroji spuštěn *netperf* v módu serveru. Na druhé je *netperf* spuštěn v módu klienta a připojuje se na server, čímž se zahajuje test. *Netperf* testy jsou dlouhé jednu minutu a zvlášť jsou testovány protokoly TCP a UDP. Po ukončení testů jsou na standardní výstup vypsány výsledky z ICMP testu a obou *netperf* testů. Nyní následují okomentované úseky kódu Python tasku, které slouží k lepšímu porozumění použití Python tasků a k popisu vlastností, které se v rámci všech testů zkoumají.

```
from lnst.Controller.Task import ctl

hostA = ctl.get_host("machine1")
hostB = ctl.get_host("machine2")
```

Zde se importuje objekt `ctl`, který slouží k ovládání testovacích strojů. Do proměnných `hostA`, respektive `hostB`, se inicializuje třída `HostAPI` pro stroje, které jsou v receptu identifikovány jako `machine1` a`machine2`.

```
hostA.sync_resources(modules=["IcmpPing", "Netperf"])
hostB.sync_resources(modules=["IcmpPing", "Netperf"])
```

Tento úsek kódu slouží k tomu, aby si oba stroje zkompilovaly zdroje potřebné k používání modulů `IcmpPing` a `Netperf`.

```
ping_mod = ctl.get_module("IcmpPing",
           options={
             "addr": hostB.get_ip("testiface", 0),
             "count": 1000,
             "interval": 0.1,
             "iface" : hostA.get_devname("testiface")})
```

Tady probíhá samotná inicializace testovacího `IcmpPing` modulu. Používá se ke zkompletování příkazu, který bude na testovaném stroji spuštěn. Kompletní výčet možných parametrů pro integrované *LNST* moduly lze vyčíst na wiki stránce projektu. V této ukázce lze vidět, že cílová adresa pingu bude adresa rozhraní `testiface` ze stroje, který je v receptu identifikován jako `machine2`. Paketů bude vysláno 1000, v intervalu 0.1s a vyslány budou z rozhraní `testiface` stroje `machine1`.

```
netserver = ctl.get_module("Netperf",
              options={
                "role" : "server",
                "bind" : hostA.get_ip("testiface")})

netperf_tcp = ctl.get_module("Netperf",
                options={
                  "role" : "client",
                  "netperf_server" : hostA.get_ip("testiface"),
                  "duration" : 60,
                  "testname" : "TCP_STREAM"})
```

To samé, ale pro testovací modul `Netperf`. Zde navíc musí být zkompilován příkaz pro *netserver*, na který se *netperf* připojuje. V tasku je obsažen ještě `UDP_STREAM` test, který zde uvedený není.

```
hostA.run(ping_mod, timeout=300)
server_proc = hostA.run(netserver, bg=True)
ctl.wait(2)
hostB.run(netperf_tcp, timeout=70)
hostB.run(netperf_udp, timeout=70)
server_proc.intr()
```

Samotné spouštění testovacích modulů probíhá následovně. Na úvod je na stroji machine1 spuštěn program *ping* s parametry, které jsou zadány při inicializaci modulu `IcmpPing`. Proměnná `timeout` je zde nastavena, aby se předešlo předčasnému ukončení testu, protože některé testy obsahují velké zpoždění, se kterým modul nepočítá a spojení po delší době neaktivity ohlásí za mrtvé a test ukončí jako neúspěšný. Po *ping* testu následuje spuštění *netserveru*. Příkaz `wait` je zde nutný kvůli době, kterou trvá než `netserver` začne naslouchat příchozím `netperf` spojením. Dále jsou spuštěny oba `netperf` testy, proměnná `timeout` je zde nastavena ze stejného důvodu jako u *ping* testu. Po ukončení testů se příkazem `intr` ukončí proces `netserver` a následuje vrácení testovacího prostředí do původního stavu a vypsání výsledků na standardní výstup.

Recepty jsou organizovány do složek, jejichž názvy korespondují s předmětem jejich testování, tzn. že ve složce `delay` jsou testy zaměřující se pouze na *NetEm* parametr delay.

Ve složce `combinations` jsou testy, ve kterých jsou *NetEm* parametry různě kombinovány. Které parametry jsou kombinovány lze vždy vyčíst z názvu souboru. Každý test má čtyři varianty: low, medium, high a extreme. Při volbě hodnot pro každou variantu jsem se snažil vybírat hodnoty tak, aby bylo pokryto co nejširší spektrum možností. *NetEm* parametry jsou nastavovány na obouch rozhraních. Po vytvoření testovací sady byly testy spuštěny na strojích firmy Red Hat pro ověření, zda-li splňují požadavky, které byly stanoveny před samotným vznikem práce. Testy byly spuštěny a ověřeny na operačních systémech Fedora 21 a Red Hat Enterprise Linux 7.

6.2 Zamyšlení

Díky testům se podařila ověřit úspěšnost implementace. Celý *tc* příkaz, který je na příslušná rozhraní aplikován, je zkompilován správně a to i v případě více zadaných *NetEm* parametrů. Kontrola validity probíhá už při parsování receptu, čímž je urychleno odhalování chyb ještě před tím, než se Slave-vové začnou konfigurovat. Pokud uživatel zadá parametry syntakticky špatně (např. zapomene znak procenta), aplikace vyvolá výjimku při konfiguraci a nekončí v nekonzistentním stavu, což je důležité především pro další spouštění testů. Konfigurace a dekonfigurace je úspěšná na strojích s *NetworkManager-em* i bez něj.

Samotná sada testů byla navrhnuta tak, pokryla hraniční stavy i stavy mezi těmi hraničními. Upravit testy pro vlastní potřeby není nic těžkého a zvládne to i člověk bez hlubších znalostí aplikace *LNST* či emulace pomocí aplikace *NetEm*. Hlavní motivací pro testovací sady je především možnost konfigurovat „špatné“ prostředí. Pokud chce vývojář/tester vyzkoušet svoji síťovou aplikaci ve ztížených podmínkách, vybere si test, který odpovídá podmínkách, které chce nasimulovat a testy spustí s přepínačem `config_only` místo přepínače `run`. Pomocí tohoto přepínače se celá síťová topologie nakonfiguruje podle zadaného XML receptu, ale žádný Python task není spuštěn. Konfigurace na stroji zůstane do té doby, dokud Slave běží nebo dokud nedostane pokyn od Controlleru-u k dekonfiguraci.

Kapitola 7

Závěr

Cílem této práce bylo nastudovat, čím se zabývá emulace sítí, seznámit se s její historií a také s alternativami, které při experimentování lze použít. Emulace je rozumným kompromisem, mezi simulací a testováním v reálném světě. Nabízí kvalitní výsledky za relativně nízké náklady. Navíc si uživatel může vybrat mezi různými typy emulátorů podle toho, který jeho potřebám vyhovuje nejvíce. V práci bylo popsáno několik emulátorů jak z kategorie softwarových, tak hardwarových. Ukázalo se, že pro účely této práce je nejvhodnějším emulátorem NetEm.

V praktické části byla úspěšně implementována podpora aplikace NetEm do aplikace LNST. Součástí implementace bylo také vytvoření testovacích sad, které pokrývají situace, jenž mohou ve WAN sítích nastat, včetně nestandardních situací, kdy se např. ztrácí velké (více než 90 %) množství paketů nebo k cíli dorazí s velkým (více než 2500 ms) zpožděním. Tyto sady slouží především pro vývojáře a testery, kteří si chtějí otestovat, jak se jejich aplikace dokáže vypořádat s degradovanou sítí. Pro vývojáře operačních systémů se celá implementace a testy hodí při testování nových verzích kernelu a ovladačů síťových karet. Mohou pomoci odhalit regrese v kódu dříve, než je kód distribuován k zákazníkům.

V případě rozšíření této práce by se mohla například přidat kontrola syntaxe ke kontrole validity parametrů. Tím by uživatel získal informaci o chybě v receptu dříve, než ji získá z výjimky, která je vyvolaná při neúspěšném spuštění zkompilovaného tc příkazu. Dále by mohla být implementována podpora NetEm parametrů limit a rate, čímž by se rozšířilo spektrum možných vlastností konfigurovatelných v testovacím prostředí. Také testovacích sad by mohlo být vytvořeno více, například na základě zpětné vazby od vývojářů síťových aplikací či vývojářů operačních systémů a ovladačů síťových karet a zařízeních.

Literatura

[1] libvirt Homepage [online]. Dostupné na http://www.libvirt.org/.

[2] netperf Homepage [online]. Dostupné na http://www.netperf.org/.

[3] PacketStorm Homepage [online]. Dostupné na http://packetstorm.com/.

[4] Ahmed Eltahir Ahmadon: *Laboratoř kvality služby*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií, 2012.

[5] Ahn, J. S.; Danzig, P. B.; Liu, Z.; aj.: Evaluation of TCP Vegas: Emulation and Experiment [online]. *SIGCOMM Comput. Commun. Rev.*, dostupné na http://doi.acm.org/10.1145/217391.217431.

[6] Antsilevich, U. J. S.; Kamp, P.-H.; Nash, A.; aj.: Dummynet [online]. Dostupné na http://info.iet.unipi.it/luigi/dummynet/.

[7] Bennett, J. C. R.; Zhang, H.: WF2Q: Worst-case Fair Weighted Fair Queueing [online]. 1996, Dostupné na http://www.cs.cmu.edu/~hzhang/papers/INFOCOM96.pdf.

[8] Carbone, M.; Rizzo, L.: Dummynet Revisited [online]. *SIGCOMM Comput. Commun. Rev.*, dostupné na http://doi.acm.org/10.1145/1764873.1764876.

[9] Hemminger, S.; aj.: Network emulation with NetEm. In *Linux conf au*, 2005, s. 18–23.

[10] Hewlett Packard: Hewelett Packard [online]. Dostupné na http://www8.hp.com/us/en/software-solutions/network-virtualization/.

[11] Jiri Pirko: LNST Project Homepage [online]. Dostupné na http://www.lnst-project.org/.

[12] Linux Foundation: NetEm Homepage [online]. Dostupné na http://www.linuxfoundation.org/collaborate/workgroups/networking/netem/.

[13] Makoto Murata: RELAX NG Homepage [online]. Dostupné na http://www.relaxng.org/.

[14] Pirko J., Tluka J., Lichtner O., Prochazka J., Pazdera R.: LNST Wiki [online]. Dostupné na https://www.github.com/jpirko/lnst/wiki.

[15] Razvan Beuran: *Introduction to network emulation*. Pan Stanford Publishing, 2013, "ISBN 978-981-4310-91-8".

Příloha A

Obsah DVD

- repo – složka s repozitářem, ve kterém byla implementace prováděna, obsahuje také soubor s licenčním ujednáním.
- tex – složka se zdrojovými kódy potřebnými pro úpravu textové části této bakalářské práce.
- prace.pdf – elektronická verze textové práce.

Příloha B

RelaxNG Schéma pro recepty

```
<define name="netem">
    <element name="netem">
        <interleave>
            <optional>
                <element name="delay">
                    <optional>
                        <ref name="options"/>
                    </optional>
                </element>
            </optional>
            <optional>
                <element name="loss">
                    <optional>
                        <ref name="options"/>
                    </optional>
                </element>
            </optional>
            <optional>
                <element name="duplication">
                    <optional>
                        <ref name="options"/>
                    </optional>
                </element>
            </optional>
            <optional>
                <element name="corrupt">
                    <optional>
                        <ref name="options"/>
                    </optional>
                </element>
            </optional>
            <optional>
                <element name="reordering">
                    <optional>
                        <ref name="options"/>
                    </optional>
                </element>
            </optional>
        </interleave>
    </element>
</define>
```