

Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Simulátor virtuální počítačové sítě Cisco

Stanislav Řehák

Vedoucí práce: Ing. Pavel Kubalík, Ph.D.

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

24. května 2010

Poděkování

Nejprve bych chtěl poděkovat své rodině, že vytvořila zázemí, které mi bylo oporou. Dále bych chtěl poděkovat vedoucímu práce Ing. Pavlovi Kubalíkovi, Ph.D. za užitečné rady a připomínky při tvorbě bakalářské práce. Mé poděkování také patří Petrovi, který byl tak hodný a smělý, že provedl uživatelské testování.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Červeném Kostelci dne 24. 5. 2010

.....

Abstract

This bachelor thesis inquires into desing and implementation of virtual network simulator composed of cisco-based routers for subject Y36PSI Počítačové sítě. Simulator allows configuration of routers via Cisco IOS command line. Simulator implements commands for configuring interfaces, static routing, dynamic and static network address translation. This system also offers verification of current configuration via ping and traceroute commands. Entire network settings are loaded from a configuration file and there is also possibility to save current configuration to a given file. User testing is integral part of a thesis.

Abstrakt

Tato práce se zabývá návrhem a implementací simulátoru virtuální počítačové sítě založené na směrovačích od firmy Cisco Systems pro předmět Y36PSI Počítačové sítě. Simulátor umožňuje konfiguraci cisco směrovače přes textové rozhraní Cisco IOS. Simulátor implementuje příkazy pro konfiguraci rozhraní, statické směrování, dynamický a statický překlad adres. Pro ověření správnosti konfigurace jsou implementovány příkazy ping a traceroute. Celá konfigurace virtuální počítačové sítě se načítá ze souboru a je možné ji následně zase uložit zpět. Součástí práce je také uživatelské testování celé aplikace.

Obsah

1	Úvod	1
2	Popis problému, specifikace cíle	3
2.1	Shrnutí funkčních požadavků	3
2.2	Nefunkční požadavky	4
2.3	Vymezení spolupráce	4
3	Existující řešení	5
3.1	Cisco Packet Tracer	5
3.2	OMNeT++ simulátor	6
3.3	Simulation Toolkit 7.0	6
3.4	Boson NetSim Network Simulator	6
3.5	Dynamips Cisco 7200 Simulator	7
3.6	Virtuální laboratoř počítačových sítí VirtLab	8
4	Analýza a návrh řešení	9
4.1	Architektura	9
4.1.1	Naivní návrh	9
4.1.2	Klient - server	10
4.1.2.1	Komunikační vrstva	10
4.1.2.2	Aplikační vrstva	10
4.1.3	Telnet	10
4.1.4	Konfigurační soubor	11
4.1.5	Směrovač	11
4.2	Podobnost simulátoru se skutečným směrovačem	11
4.3	Předpokládaný rozsah	11
4.4	Programovací jazyk a prostředí	12
4.5	Paměťová náročnost	12
4.6	Uživatelské rozhraní	12
5	Realizace	13
5.1	Komunikační vrstva	13
5.2	Aplikační vrstva	14
5.2.1	Vyhodnocování příkazů	14
5.2.2	Datové struktury jádra	15
5.2.2.1	Síťové rozhraní	15

	5.2.2.2	IP adresa	16
5.3	Parser Cisco		16
	5.3.1	Cisco IOS	16
		5.3.1.1 Uživatelský mód	16
		5.3.1.2 Privilegovaný mód	17
		5.3.1.3 Konfigurační mód	17
		5.3.1.4 Konfigurace rozhraní	18
	5.3.2	Implementace Cisco IOS	18
	5.3.3	Odchylky v implementaci	19
5.4	Načítání ze souboru		20
	5.4.1	Zpracování parametrů	20
	5.4.2	Konfigurační soubor	20
	5.4.3	Implementace SAX handleru	21
		5.4.3.1 Načítání	21
		5.4.3.2 Datová struktura	21
		5.4.3.3 Vytváření virtuálních počítačů	22
5.5	Ukládání do souboru		23
5.6	Směrování		24
	5.6.1	Debuging	24
	5.6.2	Síť č.1	24
		5.6.2.1 Popis problému	24
		5.6.2.2 Řešení	25
	5.6.3	Síť č.2	25
		5.6.3.1 Popis sítě	25
		5.6.3.2 Experimenty	26
	5.6.4	ARP protokol	27
	5.6.5	Pravidla o příjmu paketů	27
5.7	Wrapper směrovací tabulky		28
	5.7.1	Směrovací tabulka	28
		5.7.1.1 Výběr záznamů	28
	5.7.2	Wrapper	28
		5.7.2.1 Statické záznamy	28
	5.7.3	Vlastnosti	29
	5.7.4	Odchylky	31
5.8	Překlad adres		32
	5.8.1	Cisco a NAT	32
		5.8.1.1 Statický NAT	32
		5.8.1.2 Dynamický NAT a overloading	33
	5.8.2	Návrh a implementace NATu	34
		5.8.2.1 Návrh	34
		5.8.2.2 Implementace	34

6	Testování	39
6.1	Uživatelské testování	39
6.1.1	Zadání	39
6.1.2	Průběh testování	39
6.1.2.1	Úkol č.1 spuštění serveru	40
6.1.2.2	Úkol č.2 průchod stavy IOS	40
6.1.2.3	Úkol č.3 nastavení rozhraní	41
6.1.2.4	Úkol č.4 směrování	41
6.1.2.5	Úkol č.5 statický NAT	41
6.1.2.6	Úkol č.6 dynamický NAT	41
6.1.3	Shrnutí	42
6.2	Zátěžové testy	42
6.2.1	Návrh	42
6.2.2	Výsledky	42
7	Možná vylepšení	43
8	Závěr	45
A	Seznam použitých zkratk	49
B	UML diagramy	51
C	Instalační a uživatelská příručka	53
D	Obsah přiloženého CD	55

Seznam obrázků

3.1	Cisco Packet Tracer	5
3.2	OMNeT++ simulátor	6
3.3	Boson NetSim Network Simulator	7
3.4	Dynamips Cisco 7200 Simulator	7
4.1	Počáteční návrh	9
4.2	Textové rozhraní pro konfiguraci cisca	12
5.1	UML návrh komunikační části	13
5.2	Třídní model předků	14
5.3	Zjednodušený diagram tříd	15
5.4	Přehled základních módů Cisco IOS [14]	17
5.5	Síť linux - cisco	25
5.6	Síť linux1 - cisco1 - cisco2	26
5.7	Statický překlad adres	33
5.8	Návrh překladu adres č.1	34
5.9	Návrh překladu adres č.2 - finální	37
5.10	Vývojový diagram překladu adres	38
6.1	Zadání sítě pro uživatelské testování	40
D.1	Seznam přiloženého CD — příklad	55

Seznam tabulek

Kapitola 1

Úvod

Úkolem této práce je navrhnout a implementovat aplikaci, která umožní vytvoření virtuální počítačové sítě pro předmět Y36PSI Počítačové sítě. Z pohledu uživatele se systém musí tvářit jako reálná síť. Tento úkol byl rozdělen na dvě části: Cisco a Linux. Můj úkol je právě emulace Cisco IOS¹. Na dnešním virtuálním trhu existuje celá řada programů pro virtualizaci sítě. Většina z nich je však špatně dostupných (zejména kvůli licenci) nebo se nehodí pro potřeby předmětu Počítačové sítě.

Vize je taková, že student si v teple domova spustí tuto aplikaci a vyzkouší si konfiguraci na virtuálním ciscu, protože ke skutečnému nemá obyčejný student přístup. Zjistí, jak to funguje a pak už jen přijde na cvičení předmětu a vše nakonfiguruje tak, jak to má být.

Tato práce je v rozsahu týmového projektu, protože přesahuje rozsah jedné bakalářské práce. Byly vymezeny hranice, aby se tento úkol mohl rozdělit na dvě části. Nakonec celá aplikace byla rozdělena na tři části. První je část společná, kde je implementováno jádro klient - server. Druhou část tvoří Cisco IOS a právě touto částí se zabývá tato práce. A třetí část je platforma Linux, kterou zpracoval Tomáš Pitřinec v bakalářské práci Simulátor virtuální počítačové sítě Linux.

¹Internetwork Operating System je operační systém používaný na směrovačích a přepínačích firmy Cisco Systems

Kapitola 2

Popis problému, specifikace cíle

Virtuální síť musí podporovat několik k sobě připojených počítačů (Linux nebo Cisco, viz vymezení spolupráce, kapitola 2.3). Limit připojených počítačů není v zadání určen, nicméně se počítá s tím, že systém bez problému zvládne pár desítek počítačů (viz Zátěžové testy 6.2), ačkoliv v praxi jich většinou nebude potřeba více než deset. Systém musí umožnit konfiguraci rozhraní potřebných pro propojení sítě včetně příkazů pro aktivaci a deaktivaci rozhraní. Dále aplikace musí umožnit správu směrovací tabulky pomocí příkazů Cisco IOS. V předmětu Y36PSI se také požaduje po studentech, aby rozuměli překladu adres neboli tzv. „natování“ - NAT¹. Cisco podporuje hned několik druhů překladu adres. Pro tuto aplikaci je potřeba implementovat tři způsoby, které se zkoušely na cvičeních: statický překlad, dynamický překlad a dynamický překlad s metodou overloading. Dále systém musí být schopen načíst a posléze zase uložit celou konfiguraci do souboru. Funkčnost celého řešení musí být ověřitelná příkazy ping a traceroute.

2.1 Shrnutí funkčních požadavků

1. Systém musí umožnit vytvoření počítačové sítě založené na směrovačích firmy Cisco Systems.
2. Systém musí umožnit konfiguraci rozhraní.
3. Systém musí obsahovat funkční směrování.
4. Systém musí implementovat překlad adres.
5. Systém musí podporovat ukládání a načítání do/ze souboru.
6. Pro ověření správnosti musí být implementovány příkazy ping a traceroute.
7. K jednotlivým počítačům aplikace musí být umožněno připojení pomocí telnetu.
8. Pomocí telnet klientů musí být možné několikanásobné paralelní připojení k jednomu počítači najednou.

¹Network Address Translation

2.2 Nefunkční požadavky

1. Aplikace musí být multiplatformní - alespoň pro OS² Windows a Linux
2. Aplikace musí být spustitelná na běžném³ studentském počítači.
3. Systém by měl být co nejvěrnější kopií reálného Cisca v mezích zadání.

2.3 Vymezení spolupráce

Jak jsem naznačil v kapitole 1, práce je rozdělena na dvě samostatné bakalářské práce. Rozdělení dle typu počítačů na Linux a Cisco se ukázalo jako správná volba, nicméně bylo potřeba dořešit několik věcí. Hned po započatí prací jsem zjistil, že v některých věcech budu muset více spolupracovat s kolegou, protože se týkaly obou našich implementací. Např. směrovací tabulka na Linuxu a Ciscu se chová téměř úplně stejně, dokonce se podle ní i stejně směruje. Celé to má ale malý háček: Cisco má totiž tabulky dvě! První je tvořena příkazy, které zadal uživatel a druhá je vypočítávána z tabulky první. Já jsem tedy použil směrovací tabulku od kolegy, která byla primárně programována pro linux, a tu jsem zaobalil do tzv. wrapperu, který ji ovládá a sám přidává funkcionalitu tabulky druhé.

Směrování probíhá stejně na obou systémech, liší se však pravidla pro příjem paketů. Já jsem tedy pouze navázal na kolegovu implementaci tím, že jsem přidal metodu pro příjem paketů (bude detailněji vysvětleno v kapitole Směrování 5.6). Abstraktní základ příkazů `ping` a `traceroute` byl převzat od kolegy, stejně tak výpočet statistik o doručení paketu. Jelikož má Cisco překlad adres natolik robustní (rozsáhlé datové struktury), tak ho bylo možné použít s menšími úpravami⁴ pro linuxový příkaz `iptables`.

Mojí prací je také načítání a ukládání do souboru, startovací skripty pro server i klienty a uživatelská příručka. Na datových strukturách pro jádro celého systému jsem musel spolupracovat s kolegou, protože struktury musejí odrážet situaci na obou systémech. Např. třída zaštiťující IP adresu je z půlky má a z půlky kolegy. U takové třídy je vlastnictví popsáno na úrovni jednotlivých metod. U ostatních tříd je autorství uvedeno v hlavičce. Dále komunikační část je výsledkem společné práce⁵ z roku 2008, kdy jsme právě pro předmět Počítačové sítě programovali hru Karel.

²Operating System, česky operační systém

³Slovem „běžné“ se myslí v podstatě jakýkoliv počítač, na kterém je možné nainstalovat prostředí Javy - Java Runtime Environment.

⁴Stačilo přidat několik obslužných metod.

⁵Dle tehdejších pravidel jsme mohli programovací úlohy implementovat a odevzdávat ve dvojicích.

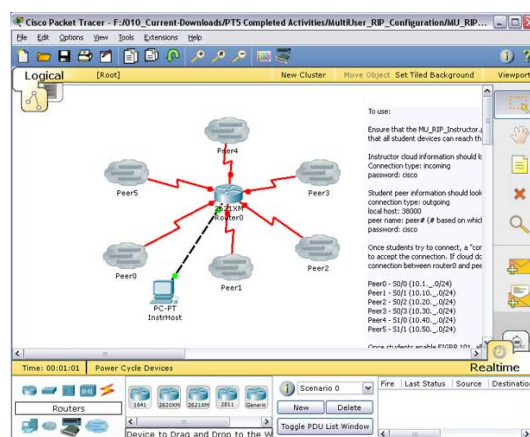
Kapitola 3

Existující řešení

Existujících implementací je na síťovém trhu celá řada, nicméně ne každé řešení je vhodné pro potřeby předmětu Y36PSI.

3.1 Cisco Packet Tracer

Asi nejznámějším simulátorem Cisca je Packet Tracer [7] od stejnojmenné firmy. Program slouží k simulaci síťového provozu počítačových sítí založených na hardware od Cisco Systems. Packet Tracer umožňuje vizualizaci provozu sítě, konfiguraci síťových prvků v grafickém i textovém módu. Program obsahuje velmi málo odchylek od skutečného cisco směrovače. Packet Tracer je dostupný pro platformy Windows i Linux zdarma pro všechny členy Cisco Networking Academy. To je zároveň velkou nevýhodou, protože licence neumožňuje jiné použití a je tedy Paket Tracer ve škole nepoužitelný.

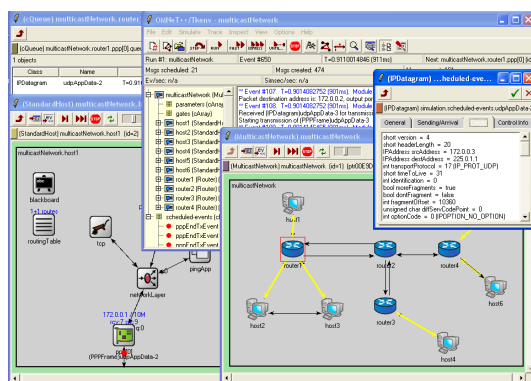


Obrázek 3.1: Cisco Packet Tracer

3.2 OMNeT++ simulátor

Simulační systém OMNeT++ [11] je velmi propracovaný opensource nástroj pro simulaci prakticky čehokoliv. OMNeT++ je postaven na modulární architektuře, takže při správných knihovnách (modulech) může simulovat počítačovou síť. Systém dokáže simulovat Cisco IOS i počítač postavený na linuxu. Aplikace je komplikovaná a představu jednoduchého programu pro výuku studentů spíše nesplňuje.

Více se tímto simulátorem zabýval Bc. Jan Michek v rámci své diplomové práce Emulátor počítačové sítě [10].



Obrázek 3.2: OMNeT++ simulátor

3.3 Simulation Toolkit 7.0

Simulation Toolkit 7.0 [19] je grafický simulátor pro testování a výuku různých síťových aplikací. Simulation Toolkit umožňuje simulaci více než 50000 SNMP služeb (v1, v2c, v3), TL1¹, TFTP², FTP³, Telnet a Cisco IOS na jediném počítači. Software je nabízen pod shareware licencí a je dostupný pro operační systémy Windows, Linux i Unix. Za poskytnutí osobních údajů lze stáhnout plně funkční zkušební verzi na 30 dní. Plná časově neomezená verze stojí od \$995 do \$14995⁴.

3.4 Boson NetSim Network Simulator

Boson NetSim Network Simulator [1] je aplikace pro simulaci síťového hardware a software a je designován jako výuková pomůcka pro začínající administrátory Cisco IOS. Systém dokáže simulovat více než 40 různých síťových prvků od firmy Cisco Systems. Simulátor obsahuje grafický i textový konfigurační režim sítě. Program je dostupný pro Windows, Linux

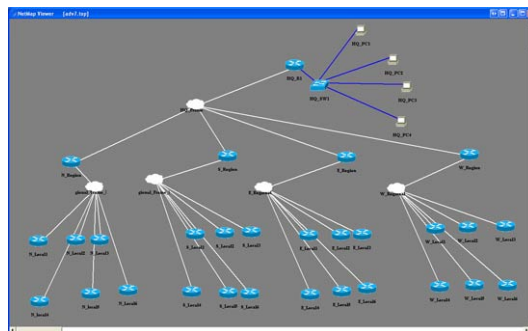
¹Transaction Language 1

²Trivial File Transfer Protocol

³File Transfer Protocol

⁴k datu 23.5.2010

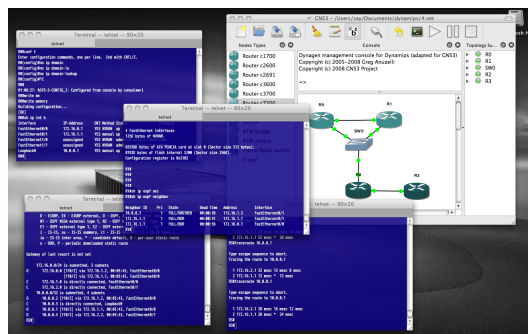
i Solaris. Podobně jako Simulation Toolkit i tento software je nabízen ve zkušební verzi na 30 dní. Plná časově neomezená verze stojí od \$99 do \$349 ⁵.



Obrázek 3.3: Boson NetSim Network Simulator

3.5 Dynamips Cisco 7200 Simulator

Dynamips Cisco 7200 Simulator [2] je program napsaný Christophe Fillot za účelem emulace Cisco směrovačů. Dynamips funguje na platformách Linux, Mac OS X, Windows a emuluje hardware Cisco směrovačů tak, že se načte obraz originálního Cisco IOS do emulátoru. Program tedy umožňuje využívat veškeré funkce Cisco IOS. Program je licencován pod GNU GPL⁶, ale obraz IOSu není bohužel volně k dispozici. Tudíž lze program legálně používat jen jako účastník kurzu CNA⁷ - to ale studenti obvykle nejsou, takže tento software také není vhodný pro výuku.



Obrázek 3.4: Dynamips Cisco 7200 Simulator

⁵k datu 23.5.2010

⁶licence pro svobodný software

⁷Cisco Networking Academy

3.6 Virtuální laboratoř počítačových sítí VirtLab

Projekt VirtLab [9] zpřístupňuje laboratorní prvky pro praktickou výuku počítačových sítí vzdáleně prostřednictvím internetu. Studenti ostravské VŠB-TU⁸ mají možnost si rezervovat pomocí webového rozhraní laboratorní síťové prvky na určitý časový interval a poté k nim přistupovat přes webový prohlížeč pomocí Java appletů. Propojení síťových prvků se provede automaticky dle zvolené úlohy. Dle mého názoru je to systém velmi ambiciózní, nicméně pro výuku Y36PSI v praxi nepoužitelný⁹.

⁸Vysoká škola báňská - Technická univerzita Ostrava

⁹Za předpokladu že ČVUT nenaváže spolupráci s VŠB-TU Ostrava.

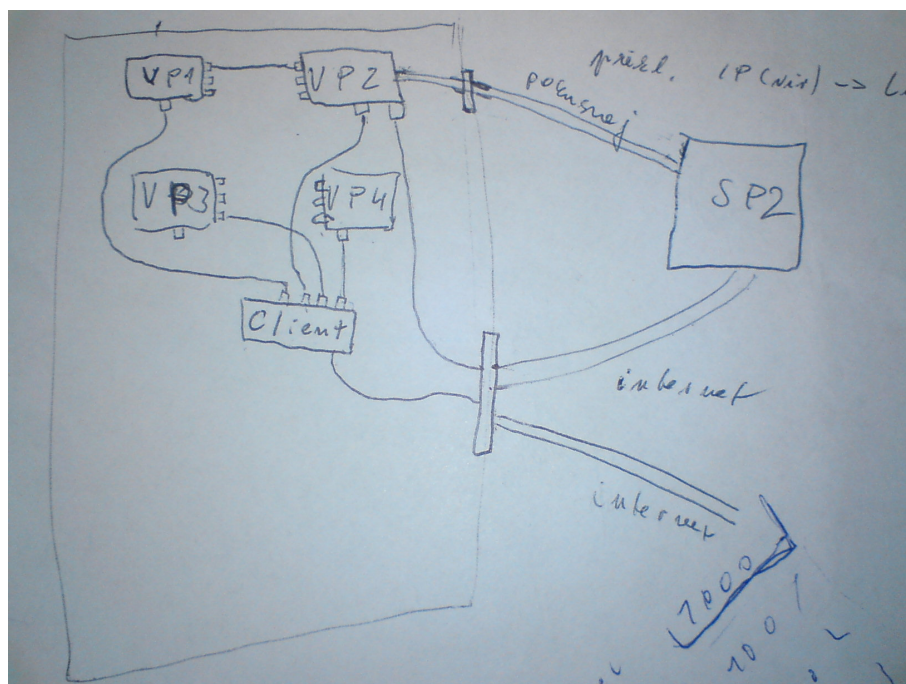
Kapitola 4

Analýza a návrh řešení

4.1 Architektura

4.1.1 Naivní návrh

Po zadání projektu jsem vytvořil počáteční návrh aplikace a po konzultaci s kolegou (každý si udělal svůj návrh) vznikl tzv. naivní návrh (viz obrázek 4.1). Tato prvotní varianta počítala s připojením do skutečné sítě, ale byla zavržena zejména kvůli složitosti takového systému a časové náročnosti implementace. Navíc propojení s reálnou sítí není mezi požadavky.



Obrázek 4.1: Počáteční návrh

4.1.2 Klient - server

Dle nového návrhu bude celá aplikace rozdělena na dvě základní vrstvy:

- komunikační - bude zajišťovat spojení mezi klientskou a serverovou částí aplikace
- aplikační - bude tvořit zbytek systému (směrování, překlad adres, parsery příkazů, ..)

4.1.2.1 Komunikační vrstva

Komunikační vrstva bude složena z architektury klient - server. Tato vrstva bude převzata ze semestrální práce pro předmět Y36PSI, kde bylo za úkol mimo jiné implementovat více-vláknový server. V této převzaté implementaci je server, ke kterému se připojují klienti. Pro každého klienta se vytvoří nové vlákno, takže není problém obsluhovat více klientů zároveň.

Cílem aplikace je vytvořit simulátor počítačové sítě, ta bude muset být reprezentována objekty, které budou zastupovat jednotlivé počítače. Každý počítač se bude muset chovat jako server, aby se na něj mohlo připojit více klientů najednou. Tak bude splněn jeden z funkčních požadavků.

Pro připojení klientů bude použit program `telnet`. Více o návrhu v kapitole 4.1.3.

4.1.2.2 Aplikační vrstva

Tuto vrstvu bude tvořit především parser příkazů emulující Cisco IOS, směrování paketů, routovací tabulku a překlad adres. Aplikační vrstva bude kvůli větší přehlednosti více rozdělena v kapitole Realizace 5.

4.1.3 Telnet

V zadání je přímo zmíněno použití programu telnet pro připojení klientů k serveru. Telnet je ale také protokol, po kterém se domlouvá telnet klient a telnet server. Česká wikipedie píše o telnet protokolu: „*Protokol přenáší osmibitové znaky oběma směry (duplexní spojení) a je velmi jednoduchý.*“ [16] Protokol tedy vše posílá po znaku a protistrana po znaku vše potvrzuje. Telnet protokol začne při navazování spojení vyjednávací proces, který vyjedná s protistranou budoucí nastavení.

Samotný telnet (ať protokol či program) ale neposkytuje doplňování příkazů nebo alespoň historii příkazů. Dalším problémem je, že při psaní příkazů přes telnet nefunguje editace aktuálního řádku, respektive lze mazat po znacích klávesou `BackSpace`, ale nelze se pohybovat do stran šipkami doleva a doprava - při takovém pokusu to vypíše `^[D` či `^[C`.

Řešením by mohlo být zkusit posílat nějaké speciální znaky pro posun kurzoru do stran, aby server věděl, že má nastat posun vlevo či vpravo. Pak by si musel server pamatovat, na které pozici řádku je klient. Nepřišel jsem však na způsob, jak přesvědčit program telnet pro posílání nějakého znaku na událost „šipka doprava“, takže tento návrh jsem musel zavrhnout.

Posun kurzoru se ale neprojevuje při připojování na vlastní telnet server. To je způsobeno tím, že v takovém případě se o editaci řádku a historii příkazů stará samotný server - v mém

případě telnet server a BASH¹. V této aplikaci toho ale nelze využít, takže musí být tyto funkcionality na straně klienta, který je bude zajišťovat.

Pro linux jsem našel program rlwrap (readline wrapper), který přidává všechny tyto užitečné funkce: editace řádky, historie příkazů, doplňování příkazů, obarvení promptu. Pro Windows jsem nic takového nenašel, takže uživatelé OS Windows budou muset tuto aplikaci spouštět přes program Cygwin. Výhodou tohoto řešení je lepší uživatelský komfort linuxové příkazové řádky než nativní Windows konzole `cmd`.

4.1.4 Konfigurační soubor

Při startu serveru by se měla načíst konfigurace ze souboru a měla by být možnost uložit aktuální konfiguraci zpět do stejného souboru. Diskuze nad možnými řešeními je popsána v kapitole 5.4.

4.1.5 Směrovač

Na skutečné počítačové síti jsou síťové prvky několika druhů (router, switch, bridge, repeater, ..), ale na laboratorních cvičeních předmětu Y36PSI se nastavují pouze směrovače ze 3. vrstvy² síťového ISO/OSI modelu. Proto tato práce bude implementovat pouze jeden typ síťového prvku - směrovač (router).

4.2 Podobnost simulátoru se skutečným směrovačem

Jedním z cílů projektu je vytvořit systém, který bude co nejvíce podobný skutečnému směrovači cisco. Bude ale potřeba položit někde hranici mezi složitostí a věrností výsledné práce, protože tyto dvě metriky jsou vzájemněm protikladu. Cisco IOS je natolik robustní a propracovaný systém, že je v mých silách implementace pouze úzké části systému, která je nutně potřeba pro splnění cíle. Pravděpodobně budu nucen místy ustoupit a nechat vypsat hlášení, že to či ono není v mé implementaci podporováno.

V samotném parseru příkazů pro Cisco IOS nebude takovéto hlášení téměř vůbec řešeno, protože by to znamenalo implementaci několika stovek pravidel pro všechny příkazy - např. příkaz `ip` má 103 možností v konfiguračním stavu. Aby ale uživatel měl alespoň nějakou možnost se dopátrat, co je či není podporováno, tak bude přidán příkaz `help` (`help_en` pro výpis v angličtině), který bude popisovat, jaký příkaz lze v kterém stavu Cisca použít.

4.3 Předpokládaný rozsah

Celý projekt by měl být v rozsahu zhruba 10000 řádků kódu. Nejrozsáhlejší třídy budou pravděpodobně datové struktury pro všechny objekty, které budou emulovat skutečné počítače s rozhraními, routovací a NAT tabulku atd.

¹Bourne again shell - nejpoužívanější unixový shell

²Tato „síťová vrstva“ se stará o směrování v síti a síťové adresování. Dále poskytuje spojení mezi vzdálenými sítěmi, které spolu přímo nesousedí.

4.4 Programovací jazyk a prostředí

Pro implementaci simulátoru jsem si zvolil programovací jazyk Java hned z několika důvodů. Jazyk je to velmi robustní s bohatou sadou různých knihoven. Navíc programy vytvořené v tomto jazyce jsou zpravidla jednoduše přenositelné mezi různými operačními systémy, což je jeden z bodů nefunkčních požadavků. Jazyk Java disponuje propracovaným systémem výjimek, takže při nějaké neočekávané chybě se dozvím víc, než v jazyce C++ s jeho `Segmentation fault`. Neméně významným důvodem je i skutečnost, že s Javou mám zatím největší zkušenosti.

Celá práce bude implementována v Netbeans IDE³ verze 6.8.

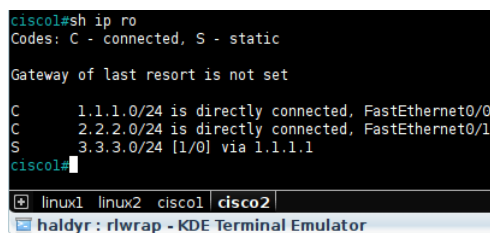
4.5 Paměťová náročnost

Paměťová náročnost bude pravděpodobně vyšší než při použití jazyka C++, jelikož javovský garbage collector není úplně ideálním řešením pro uvolňování paměti. Nicméně se neočekává použití tohoto simulátoru pro více než pár desítek virtuálních počítačů, takže spotřeba paměti by neměla překročit požadavky pro běžné aplikace v Javě. Zaplnění paměti touto aplikací by mohlo být kolem odhadem 10-30MB (+ načtené prostředí JRE).

4.6 Uživatelské rozhraní

Uživatelské rozhraní by mělo být v zásadě velmi jednoduché. Vše bude ovládáno přes příkazovou řádku, tak jako na skutečném ciscu. Spuštění serveru bude ulehčeno pomocným skriptem `start_server.sh`, který bude zároveň obsahovat nápovědu. Pro připojování klientů budou rovněž připraveny skripty.

Na obrázku 4.2 je příklad, jak by mohlo uživatelské rozhraní vypadat pod operačním systémem Linux.



```
cisc01#sh ip ro
Codes: C - connected, S - static

Gateway of last resort is not set

C      1.1.1.0/24 is directly connected, FastEthernet0/0
C      2.2.2.0/24 is directly connected, FastEthernet0/1
S      3.3.3.0/24 [1/0] via 1.1.1.1
cisc01#
```

Obrázek 4.2: Textové rozhraní pro konfiguraci cisca

³Integrated Development Environment

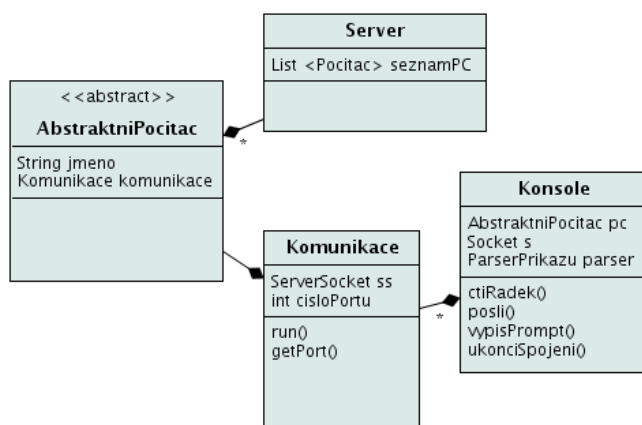
Kapitola 5

Realizace

Téměř všechny části systému obsahují tzv. debugovací mód, který vypisuje extra informace, co se právě děje nebo alepsoň přidává další vlastnost vhodnou pro ladění.

5.1 Komunikační vrstva

Server má sám pro sebe vlastní vlákno ve kterém běží. Dále server vytvoří při startu pro všechny počítače nová vlákna, která poslouchají na portu o jedna větším než předchozí počítač (první počítač začíná na portu předaným jako parametr při startu serveru). Tyto vlákna se chovají zase jako servery. Když se uživatel připojí na libovolný počítač, tak se vytvoří další vlákno pro obsluhu tohoto klienta. Výhodou tohoto řešení je, že je možné se připojit na kterýkoliv počítač kolikrát potřebujeme. Je to tedy přesně tak, jako bychom se připojovali na reálné Cisco či linux např. přes protokol ssh¹ či telnet.



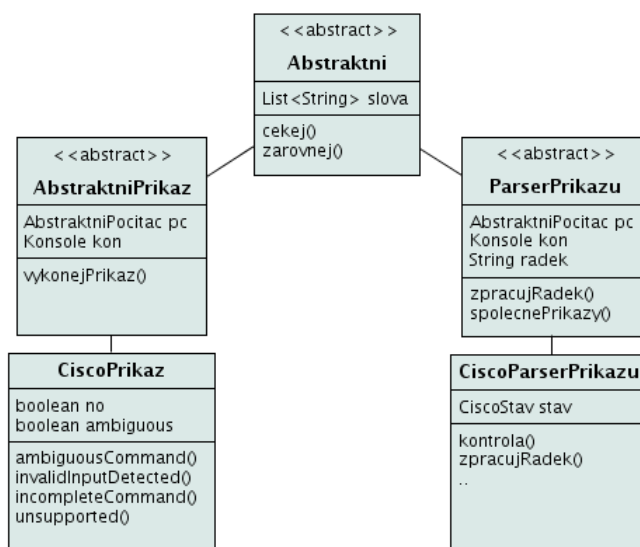
Obrázek 5.1: UML návrh komunikační části

¹Secure Shell - zabezpečený komunikační protokol (v současné době náhrada telnetu)

Na obrázku 5.1 je znázorněna komunikační část pomocí UML² diagramu. Každý počítač má objekt *Komunikace*, ta čeká na připojení nového klienta. Když klient vyšle požadavek o nové spojení, tak se vytvoří *Konsole*, která tohoto klienta bude obsluhovat. Při odpojení klienta *Konsole* zaniká, protože její přítomnost už není potřeba. *Komunikace* se tedy chová jako server, který vytváří klienty - *Konsole*.

5.2 Aplikační vrstva

Předkem všech tříd systému na aplikační vrstvě je *Abstraktni*, která zastřešuje převážně statické funkce např. *zaokrouhli* (na tři desetinná místa), *cekej* (metoda pro uspání vlákna, užitečná při výpisech Cisco IOS) a další. Od *Abstraktni* dědí abstraktní *ParserPrikazu*, který seskupuje všechny parsery příkazů (v současné době pro linux a Cisco IOS). Společný předek linux a cisco příkazů je *AbstraktniPrikaz*, z kterého dědí linuxové příkazy kolegy a hlavně *CiscoPrikaz*, který ještě přidává užitečné metody speciálně pro cisco příkazy. Viz obrázek 5.2.



Obrázek 5.2: Třídní model předků

5.2.1 Vyhodnocování příkazů

Když uživatel zadá příkaz a ukončí ho znakem nového řádku (klávesa Enter, znak `\n`), tak se ve třídě *Konzole* zavolá metoda *zpracujRadek()*. Tuto metodu vlastní abstraktní *ParserPrikazu*, který je v mém případě implementován jako *CiscoParserPrikazu*³. Ten se

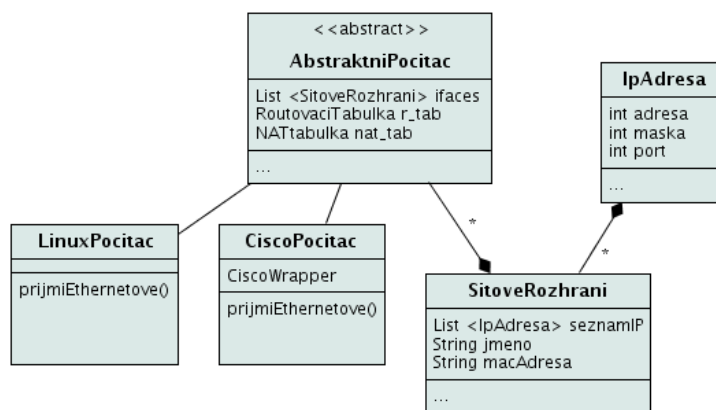
²Unified Modeling Language, UML je v softwarovém inženýrství grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.[17]

³LinuxParser příkazů má na starosti kolega. Pro jiné typy počítačů je nutno implementovat parser vlastní.

stará o zpracování poslané řádky a podle toho volá různé Cisco příkazy, které tvoří IOS, nebo jiné servisní (obslužné) příkazy.

5.2.2 Datové struktury jádra

Datové struktury pro počítačovou síť byly vytvářeny ve spolupráci s kolegou. `CiscoPocitac` je má práce, třídy `IpAdresa` a `SitoveRozhrani` obsahují metody ode mne i od kolegy. Vlastnictví je blíže popsáno u jednotlivých metod.



Obrázek 5.3: Zjednodušený diagram tříd

5.2.2.1 Síťové rozhraní

Datová struktura pro síťové rozhraní je ve své podstatě jednoduchá. Obsahuje jméno, seznam IP adres přiřazených k tomuto rozhraní, MAC⁴ adresu a stav.

Systémem je oficiálně podporována pouze jedna IP adresa per rozhraní, více adres si ale vyžádal překlad adres. MAC adresa je v tomto systému spíše pro větší přiblížení skutečnému rozhraní, protože ARP⁵ protokol v této aplikaci přímo implementován. Systém obsahuje pouze několik pravidel, které byly nutné pro rozhodování zda přijmout či nepřijmout příchozí paket. Dále rozhraní obsahuje indikátor stavu, ve kterém se nachází - zapnuté/vypnuté. Rozhraní cisca jsou ve výchozím stavu vypnutá.

⁴MAC - Media Access Control, je fyzická adresa, kterou používá 2. (spojová) vrstva ISO/OSI modelu

⁵„Address Resolution Protocol se v počítačových sítích s IP protokolem používá k získání ethernetové MAC adresy sousedního stroje z jeho IP adresy. Používá se v situaci, kdy je třeba odeslat IP datagram na adresu ležící ve stejné podsíti jako odesílatel. Data se tedy mají poslat přímo adresátovi, u něhož však odesílatel zná pouze IP adresu. Pro odeslání prostřednictvím např. Ethernetu ale potřebuje znát cílovou ethernetovou adresu.“[12]

5.2.2.2 IP adresa

`IpAdresa` je mnohem složitější třída než rozhraní i když obsahuje pouze tři čísla reprezentující adresu, masku a port. Složitost je dána tím, že tato třída obsahuje přes 40 obslužných metod, které pokrývají veškerou práci, kterou je potřeba vykonávat (výpočet čísla sítě a broadcastu, kontrolní metody pro ověřování správnosti IP adresy, ..).

5.3 Parser Cisco

5.3.1 Cisco IOS

Cisco IOS je operační systém, který se nachází na drtivé většině směrovačů firmy Cisco Systems. IOS obsahuje pouze ovládání přes příkazový řádek (CLI⁶). IOS obsahuje tzv. zkracování příkazů, které zefektivňuje práci s celým systémem. Celé to funguje tak, že když uživatelův příkaz lze doplnit na jeden jedinečný příkaz (samotné doplnění přes klávesu TAB), tak to takový příkaz hned zavolá. Například příkaz `sh run` lze jednoznačně doplnit na `show running-config`, ale kratší `sh ru` už ne:

```
Router#sh ru?  
rudpv1  running-config
```

IOS tvoří několik stavů, např.:

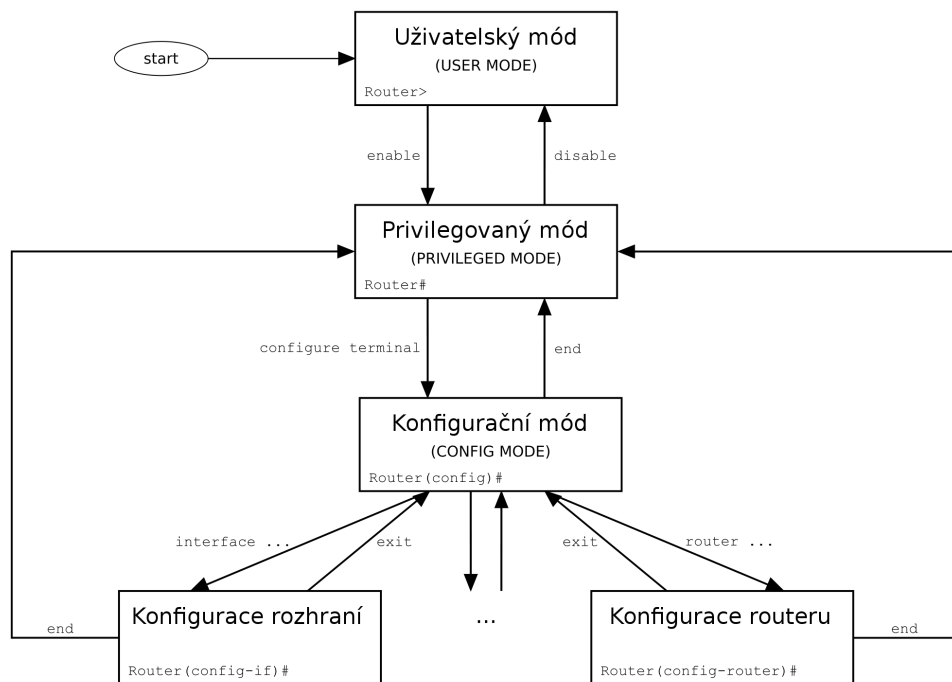
- uživatelský mód
- privilegovaný mód
- konfigurační mód - zde se nastavují volby, které ovlivní celý systém
- konfigurace rozhraní - konfigurace jednoho určitého rozhraní

Na obrázku 5.4 jsou zobrazeny důležité stavy IOS a přechody mezi nimi.

5.3.1.1 Uživatelský mód

Uživatelský mód (USER MODE) je výchozí (startovací) mód. Tento mód je značně limitovaný a dovoluje použití čistě read-only příkazů (tj. takových, které nezmění konfiguraci). Přesto má tento mód svoje opodstatnění, dovoluje např. vypsát směrovací tabulku (`show ip route`) či použít příkazy `ping` nebo `traceroute`. Do privilegovaného režimu se lze přepnout příkazem `enable`.

⁶Command Line Interface



Obrázek 5.4: Přehled základních módů Cisco IOS [14]

5.3.1.2 Privilegovaný mód

Privilegovaný mód (PRIVILEGED MODE) nebo také „administrátorský“ mód je podobný linuxovému `root` účtu. Tento mód je výchozím bodem pro vstup do ostatních módů. Pro návrat zpět do uživatelského režimu existuje příkaz `disable`. Příkaz `configure` způsobí přepnutí do dalšího konfiguračního módu. Privilegovaný mód umožňuje vypsat veškeré informace o aktuální konfiguraci systému, např.:

- `show running-config` - shrnutí aktuální konfigurace
- `show ip route` - výpis směrovací tabulky
- `show ip nat translations` - výpis dynamických záznamů v NAT tabulce

Není důvod, proč by v tomto stavu nefungovaly příkazy `ping` a `traceroute`.

5.3.1.3 Konfigurační mód

Konfigurační mód (CONFIG MODE) je jeden z nejdůležitějších, protože umožňuje konfiguraci směrovacích záznamů (`ip route`), přístupových seznamů pro potřeby překladu adres (`access-list`), pooly IP adres (`ip nat pool`) a výběr rozhraní pro přechod do stavu konfigurace rozhraní (`interface`).

5.3.1.4 Konfigurace rozhraní

V tomto módu lze nastavovat IP adresy na aktuálně vybrané rozhraní, nastavovat příznaky pro veřejné a soukromé rozhraní pro NAT (`nat inside`, `nat outside`) nebo také zapínat či vypínat rozhraní. Pro přechod ze všech konfiguračních módů do privilegovaného stačí napsat příkaz `end` nebo jen stisknout klávesovou zkratku `Ctrl+Z`.

5.3.2 Implementace Cisco IOS

Cisco IOS obsahuje desítky příkazů z nichž každý může mít až stovky variací. Proto jsem implementoval pouze úzkou část příkazů, která je potřebná pro splnění zadání této práce. Nejdůležitější funkcí parseru je rozpoznávání zkrácených příkazů. Na skutečném Cisco se opravdu procházejí všechny možnosti, které mohou v daném stavu nastat, a podle nich probíhá vyhodnocování. V mé implementaci ale mám pouze část příkazů, takže jsem to musel vyřešit jiným způsobem. Pro každé slovo (část příkazu) si `CiscoParserPrikazu` drží počet písmen, který je potřeba k jednoznačnému určení příkazu. Tato čísla jsem „naměřil“ na školních ciscích v březnu 2010. Zajímavé je, že už o 2 měsíce později jsem objevil drobné změny. Čísla se mohou měnit s různými verzemi Cisco IOS. To bych ale neviděl jako zásadní problém. Většina studentů (alespoň dle mé zkušenosti) stejně píše celé příkazy a zkrácené verze nepoužívá, protože je zpravidla nezná.

Vyhodnocování příkazů zajišťuje metoda `kontrola(command, cmd)`. Parametr `command` je celý příkaz, na který by se to mohlo eventuálně doplnit, a `cmd` je příkaz poslaný od uživatele. Nejdříve se zjistí počet znaků, který je potřeba pro jednoznačné doplnění na příkaz `command`. Po té se zkontroluje požadovaný počet znaků a také jestli zkrácený příkaz odpovídá doplněnému. A takto to vypadá v kódu:

```
if (cmd.length() >= i && command.startsWith(cmd)) {
    // i..požadovaný počet znaků
    // lze doplnit na jeden jedinečný příkaz
    return true;
}
if (command.startsWith(cmd)) {
    // vypsát amiguous command
    nepokracovat = true;
}
```

Jednotlivé příkazy Cisco IOS jsou implementovány v samostatných třídách. Třída `CiscoParserPrikazu` zajišťuje přechody mezi stavy (módy) a navíc se stará o „nahazování“ rozhraní. Přepnutí stavu rozhraní je natolik triviální, že se by se nevyplatilo mít pro to zvláštní třídu.

Ladící mód zjednodušuje testování parseru a přidává tyto funkce:

- klávesa `Enter` funguje jako přechod z uživatelského do privilegovaného módu

- použití příkazů z jiných módů v privilegovaném módu - navíc např. `ip route`, `ip nat pool inside`, `access-list`, ..
- extra výpis dynamických záznamů v natovací tabulce
- výpis `show running-config` je pro přehlednost zkrácen
- možnost testování routovací tabulky přes linuxový příkaz `route`
- používání linuxového příkazu `ifconfig`

Použití těchto funkcí je vhodné spíše pro ladění programu do budoucna než pro běh v „ostrém“ provozu. Tento mód je ve výchozím stavu vypnut.

5.3.3 Odchytky v implementaci

Má implementace Cisco IOS má navíc pár příkazů, které jsou potřeba pro ovládání systému. Jak už jsem se zmiňoval v kapitole 4.2 je zde navíc `help` a `help_en` pro výpis nápovědy. Příkaz `kill` přijde vhod, když uživatel chce ihned vypnout aplikaci a nechce projít přes několik stavů příkazem `exit`. Další servisní příkaz je `save` nebo také `uloz`, který zapíše aktuální konfiguraci všech počítačů do konfiguračního souboru, se kterým byl spuštěn nebo který byl předán jako parametr. Dále lze využít velmi jednoduchý příkaz `?` (otazník), který vypíše seznam dostupných příkazů v aktuálním stavu.

Na skutečném Ciscu funguje kombinace kláves `Ctrl+Z` pro přechod do privilegovaného módu. Ale kvůli použití programu `rlwrap` je systém limitován. Omezení spočívá v tom, že `rlwrap` přepošle signál operačnímu systému a ten pozastaví tento proces. Proces lze obnovit příkazem `fg` (na OS Linux), bohužel klientský program `telnet` neumí po pozastavení obnovit svoji funkčnost a přestává posílat vstup na standartní výstup. Je tedy už nepoužitelný a pro tyto případy existuje příkaz `kill`, který ukončí tento rozbitý proces, aby se klient mohl připojit znovu. Lépe je na tom klávesová zkratka `Ctrl+C`, která pouze ukončuje (signál `SIG_INT`) daný proces a tedy funguje bez problému.

Program `rlwrap` je ale šířen jako balíček pod programem `cygwin` v zastaralé verzi⁷, která neumožňuje přeposílání signálů, takže při stisku `Ctrl+C` i `Ctrl+Z` přestane `telnet` klient vypisovat na standartní výstup a nezbyvá než použít `kill`. Toto platí pouze na OS Windows, pro linux jsou dostupné nejnovější balíčky v repozitářích každé distribuce.

⁷ stav z 18.5.2010

5.4 Načítání ze souboru

5.4.1 Zpracování parametrů

Po spuštění startovacího skriptu `start_server.sh` se nejdříve zpracují všechny parametry. Když je nalezen parametr `-n`, tak se načítá z konfiguračního souboru pouze kostra sítě s počítačema a rozhraníma bez jejich nastavení. Pozice tohoto parametru není důležitá, systém nejprve detekuje přítomnost tohoto parametru a pak už ho ignoruje. Další parametr je název konfiguračního souboru, ten může být zadán bez koncovky `.xml`, nejdříve se zkusí načíst soubor bez koncovky a když takový soubor není, tak se načte soubor s koncovkou. Třetím parametrem je port, od kterého budou poslouchat jednotlivé počítače. Port je nepovinný, ve výchozím stavu je to port 4000. Když je daný port obsazený, tak se program ukončí z chybovou hláškou.

5.4.2 Konfigurační soubor

Nejdříve bylo nutno rozhodnout podobu výsledného konfiguračního souboru. Existuje několik možností, např.:

- javovské **Properties** ve stylu „proměnná = hodnota“, moc se nehodí na velmi členité struktury

```
compile.on.save=false
do.depend=false
do.jar=true
javac.debug=true
```

- podoba konfiguračních souborů KDE⁸, kde jednotlivé sekce jsou odděleny jménem v hranatých závorkách

```
[Xdmcp]
Enable=false
[Shutdown]
# Default is "/sbin/halt"
HaltCmd=/sbin/shutdown
```

- XML⁹ soubor, který umožňuje libovolnou strukturu

Já jsem si vybral technologii XML pro jeho robustnost a velmi dobrou čitelnost.

Načítání z XML souboru lze udělat minimálně dvěma způsoby: Vztít cizí knihovnu, která tuto funkcionalitu zajišťuje, nebo vytvořit vlastní třídu. Obě možnosti mají své výhody i nevýhody. Cizí knihovna by mohla být téměř bez práce a pravděpodobně by podporovala i zpětné ukládání. Na druhou stranu by byla malá možnost ovlivnění výsledného výstupu a bylo by vše na té knihovně závislé. Navíc s novými verzemi by se mohla měnit i její funkčnost. Znamenalo by to také instalaci této knihovny na uživatelských počítačích. Z těchto důvodů jsem zvolil druhou variantu: vlastní implementace zpracování XML.

⁸K Desktop Environment je desktopové prostředí pro Linux a další unixové operační systémy.

⁹Extensible Markup Language je rozšiřitelný značkovací jazyk

5.4.3 Implementace SAX handleru

Zpracovat XML lze přes technologii DOM¹⁰ a SAX¹¹. Já se rozhodl pro SAX z těchto důvodů:

- jednorázové sekvenční čtení
- menší paměťová náročnost
- oproti DOMu i několikrát rychlejší, což by mohlo být znatelné u velmi rozlehlé sítě

Můj `SAXHandler` tvoří tři části: samotné načítání, datová struktura pro počítač a vytváření virtuálního počítače.

5.4.3.1 Načítání

`ContentHandler` „vyhazuje“ události při zpracování XML souboru. `SAXHandler` musí tyto události odchyťovat a pokud je to událost, která nás zajímá, tak se zpracuje tj. uloží do datové struktury. `SAXHandler` musí implementovat metody na zpracování začátku elementu, konce elementu, znaková data a konce dokumentu. Po načtení konce elementu se vytvoří virtuální síť počítačů. Pro správnou funkci `SAXHandler` je důležité mít ve složce s konfiguračními soubory také soubor DTD¹², který definuje strukturu XML dokumentu.

5.4.3.2 Datová struktura

Pro ukládání informací slouží datová struktura `PocitacBuilder`, která si drží veškeré informace načtené z XML souboru o jednom počítači:

- jméno a typ počítače
- nastavení rozhraní - jméno, adresa, maska, stav
- routovací tabulka - výčet záznamů
- `ip_forward` - pro potřeby OS linux
- překlad adres - pooly adres, access-listy, přiřazení access-listů k poolům, statické záznamy

Pak je tu ještě sekundární struktura pro uložení kabelů k jednotlivým počítačům.

¹⁰Document Object Model

¹¹Simple API for XML

¹²Document Type Definition

5.4.3.3 Vytváření virtuálních počítačů

Po vyhození události konec souboru se začnou vyrábět virtuální počítače. Pokud byl použit parametr `-n`, tak se nejdříve smažou nastavení, která nemají být načtena. Po té se postupně budou načítat (a kontrolovat) všechna uložená nastavení. V zásadě lze říci, že když systém narazí na neplatná data v konfiguračním souboru, tak vypíše chybovou hlášku na standardní chybový výstup. Pokud je to chyba zásadní, tak se vyhodí výjimka, vypíše hláška a celý server se ukončí, protože nemůže pokračovat v další činnosti. Slovem zásadní je myšleno např. chybějící jméno rozhraní (kabely v XML jsou napojeny přes jména rozhraní), natažené kabely a opakující se jména počítačů či rozhraní na jednom počítači. Kabely jsou natolik klíčovou věcí, že uživatele upozorní na chybu pádem programu s výpisem, co je špatně.

Takto vypadá příklad konfiguračního souboru:

```
<pocitac jmeno="cisco1" typ="cisco">
  <rozhrani>
    <jmeno>FastEthernet0/0</jmeno>
    <ip>192.168.1.254</ip>
    <maska>255.255.255.0</maska>
    <mac>00:0b:0c:0d:0a:01</mac>
    <nahozene>true</nahozene>
    <nat>soukrome</nat>
  </rozhrani>
```

Nastavení propojení mezi počítači (respektive jejich rozhraními) je dáno v konfiguračním souboru. Původně každé rozhraní obsahovalo informaci, ke kterému rozhraní kterého počítače je připojeno. To se ale ukázalo jako zbytečně matoucí, protože informace o kabelu tam byla obsažena dvakrát. V dnešní verzi programu je to zjednodušené tak, že samotné rozhraní nenese žádnou informaci o kabelu. Kabely jsou v konfiguračním souboru zvlášť. Tento příklad ukazuje propojení třech počítačů:

```
<kabelaz>
  <kabel>
    <prvni>linux1:eth0</prvni>
    <druhy>linux2:eth0</druhy>
  </kabel>
  <kabel>
    <prvni>linux2:eth1</prvni>
    <druhy>cisco1:FastEthernet0/0</druhy>
  </kabel>
  <kabel>
    <prvni>cisco1:FastEthernet0/1</prvni>
    <druhy>cisco2:FastEthernet0/0</druhy>
  </kabel>
</kabelaz>
```

Konce kabelu jsou charakterizovány dvěma záznamy, každý obsahuje jméno počítače a rozhraní oddělené dvojtečkou.

5.5 Ukládání do souboru

Abstraktní `ParserPrikazu` obsahuje metodu, do které jsou vloženy společné příkazy pro všechny operační systémy. V současné době je tam pouze příkaz `uloz` alias `save`. Uživatel může použít jakoukoliv variantu dle libosti. Při zavolání tohoto příkazu bez parametru se bude ukládat do stejného souboru, ze kterého se při staru aplikace načítalo. Nebo může uživatel specifikovat jméno souboru (včetně cesty), do kterého se má aktuální konfigurace uložit.

Ukládání do souboru je realizováno čistě textově, tzn. vše se posílá přes `BufferedWriter` bez použití externích knihoven. Pro usnadnění práce jsem si napsal několik pomocných metod, kde např. pro uložení MAC adresy do XML stačí zavolat `zapisElement("mac", rozhrani.macAdresa)`. Velmi užitečná metoda je také `vratElement`, která postaví element s daným jménem a obsahem:

```
private String vratElement(String jmeno, String obsah) {
    if (obsah == null) {
        obsah = "";
    }
    return "<" + jmeno + ">" + obsah + "</" + jmeno + ">\n";
}
```

Výhodou tohoto řešení je maximální kontrola nad výstupem příkazu `save` a jednoduchost této implementace. Mezi nevýhody bych uvedl hlavně změnu v datových strukturách. Když by bylo potřeba připsat novou volbu, která by se měla ukládat do XML souboru, tak je nutné přidat pravidla pro načítání z XML v `SAXHandler` a navíc zde pro ukládání.

5.6 Směrování

Směrování implementoval kolega, nicméně Cisco se nechová vždy stejně, a tak bylo nutné vyčlenit rozhodování o příjmu paketů do koncových počítačů a implementovat je každý zvlášť. Nejsložitější bylo zjistit, jak se cisco přesně chová a proč. Všechny vyzkoumané informace byly platné v období březen - duben 2010. Někdo předělával školní cisca po tom, co jsem prováděl experimenty, takže je možné, že některé věci se mohou chovat jinak (nové verze IOS či jen jiná konfigurace).

Při různých experimentech jsem zjistil, že když linuxu přijde paket, na který nemá záznam ve směrovací tabulce, tak pošle zpátky patek **Destination Network Unreachable**, zatímco školní cisca posílají **Destination Host Unreachable**.

Při testování standardních případů je vše jasné, ale když jsem zkusil nakonfigurovat síť trochu neobvykle, tak to tak jasné nebylo. Experimenty byly ověřovány pomocí příkazu `ping`, který implementoval kolega¹³.

5.6.1 Debuging

Na stránkách firmy Cisco Systems jsem objevil několik návodů týkajících se vypisování zpracování paketů na jednotlivých strojích. Bez těchto návodů lze jen těžko hádat, které pakety se posílají kam. Velmi užitečné jsou příkazy, které poskytují detailní výpis aktuálně zpracovávaných paketů:

- `debug ip packet detail` - IP pakety
- `debug ip icmp` - ICMP¹⁴ pakety
- `debug arp` - ARP pakety

5.6.2 Síť č.1

5.6.2.1 Popis problému

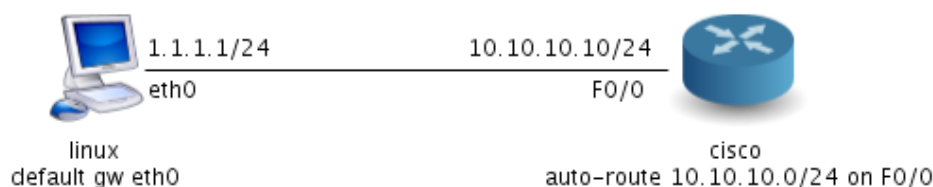
Síť na obrázku 5.5 je složená pouze ze dvou počítačů cisco a linux. Tyto počítače mají nastavené IP adresy z jiných sítí, linux má nastavený defaultní záznam na rozhraní `eth0` a cisco má samo-nastavující se záznam na síť, která je připojena na rozhraní `F0/0`¹⁵. Samo-nastavující znamená, že cisco přidává záznamy do routovací tabulky podle IP adresy z jeho rozhraní. Tuto funkcionalitu lze vypnout, na školních ciscích je ale defaultně zapnutá. Cisco nepřidává tento záznam v případě, že na druhém konci kabelu buď nikdo není nebo je rozhraní shozené (vypnuté).

Při připojení na linux a spuštění příkazu `ping 10.10.10.10` se stane, že u prvních pár paketů (při mém testování to bylo 9) vyprší timeout a pak už linux sám sobě vypisuje

¹³viz kapitola vymezení spolupráce 2.3

¹⁴Internet Control Message Protocol je jeden z nejdůležitějších protokolů ze sady protokolů internetu. Používají ho operační systémy počítačů v síti pro odesílání chybových zpráv, například pro oznámení, že požadovaná služba není dostupná nebo že potřebný počítač nebo router není dosažitelný. [13]

¹⁵Na obrázcích sítí se vyskytují zkratky `F0/0` a `F0/1`, které značí `FastEthernet0/0` resp. `FastEthernet0/1`



Obrázek 5.5: Síť linux - cisco

Destination Host Unreachable. Dlouho jsem se snažil přijít na to, proč to tak je. Těžko jsem zjišťoval, co se děje, protože jsem neměl žádné zkušenosti se sledováním paketů přes cisco směrovače.

5.6.2.2 Řešení

Nejdříve vysvětlím proč prvních několik paketů prošlo až na cisco a na další `icmp_req` odpověděl linux sám sobě. Je to způsobeno tím, že při prvních paketech ještě cisco nevědělo co s těmi pakety bude, a tak je přijalo, aby mohlo vyhodnotit co dál. Cisco ale hned přišlo na to, že nemá žádný záznam na `1.1.1.1`, takže neví, co s takovými pakety dělat, tak se radši rozhodlo, že je ani nepřijme. Obvykle cisco nepřijímá pakety hned od začátku, když nemá záznam pro takový paket, ale školní cisco mají starší verzi software a celkově reagují trochu pomaleji, než je zvykem, takže to moho být způsobeno tím.

Průchod ARP žádostí

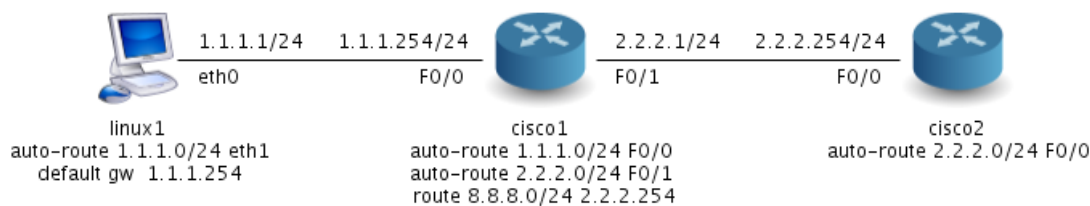
Linux nejprve pošle ARP request, aby zjistil MAC adresu cisco. Cisco přijme ARP request a snaží se odpovědět (poslat ARP reply). Problém je, že nemá záznam v routovací tabulce na adresu `1.1.1.1`, takže ani neodpoví na ARP request, tak „hezky“ je nastavené cisco.

Po přidání záznamu `ip route 1.1.1.0 255.255.255.0 F0/0` do routovací tabulky na směrovači cisco síť funguje bez jakýchkoliv problémů. Do této chvíle jsem netušil, že je něco takového možné. A ono stačí správně nastavit směrovací záznamy na obou počítačích.

5.6.3 Síť č.2

5.6.3.1 Popis sítě

Na další síti 5.6 jsou počítače linux1, cisco1 a cisco2 zapojené v jednom řetězu. Konfigurace rozhraní a směrovacích tabulek je vepsána v obrázku. Linux1 má teoreticky v dosahu (přes defaultní záznam) cisco2, to mu ale nemůže odpovědět, protože pro linux1 nemá záznam. Cisco1 přeposílá pakety z cílovou adresu `8.8.8.0/24` na bránu - cisco2.



Obrázek 5.6: Síť linux1 - cisco1 - cisco2

5.6.3.2 Experimenty

I. experiment

První experiment je `ping` z linux1 směrem na cisco2:

```
linux1:/home/dsn# ping -c2 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1006ms
```

Vše dopadlo dle očekávání tedy paket proplul až na vzdálené cisco2, které nemělo pravidlo pro manipulaci paketů s cílem 8.8.8.8, a tak chtělo poslat zpátky odesílateli `Destination Host Unreachable`. Problém je, že cisco2 nemá záznam ve směrovací tabulce pro zdrojovou adresu 1.1.1.1 (linux1), takže vyprší timeout, protože nelze doručit odpověď o nedoručení.

Po přidání záznamu pro obsluhu sítě 1.1.1.0/24 na bránu 2.2.2.1 na směrovači cisco2, tak vše už funguje dle očekávání:

```
linux1:/home/dsn# ping -c2 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
From 2.2.2.254 icmp_seq=1 Destination Host Unreachable
From 2.2.2.254 icmp_seq=2 Destination Host Unreachable
```

II. experiment

Pokud zkusíme odeslat `ping` na 1.1.1.2, což je adresa v síti mezi linux1 a cisco1, ale není to adresa ani jednoho z nás, tak dopadne takto:

```
linux1:/home/dsn# ping -c2 1.1.1.2
PING 1.1.1.2 (1.1.1.2) 56(84) bytes of data.
From 1.1.1.1 icmp_seq=1 Destination Host Unreachable
From 1.1.1.1 icmp_seq=2 Destination Host Unreachable
```

Linux ví (díky ARP protokolu), že vedle něj není počítač s IP adresou 1.1.1.2, a tak se to ani nesnaží odeslat a odpovídá sám sobě, že je host nedostupný.

III. experiment

Cisco má trochu odlišné chování:

```
cisco1#ping 1.1.1.2
```

```
Type escape sequence to abort.
```

```
Sending 5, 100-byte ICMP Echos to 1.1.1.2, timeout is 2 seconds:
```

```
.....
```

```
Success rate is 0 percent (0/5)
```

Cisco se zeptalo ethernetově (přes ARP) linuxu, ten odpověděl, že takovou adresu nemá a cisco vypsaló '.', což znamená, že vypršel timeout (dle [4]). Zatímco linux poslal DHU¹⁶.

5.6.4 ARP protokol

„Address Resolution Protocol (ARP) se v počítačových sítích s IP protokolem používá k získání ethernetové MAC adresy sousedního stroje z jeho IP adresy. Používá se v situaci, kdy je třeba odeslat IP datagram na adresu ležící ve stejné podsíti jako odesílatel. Data se tedy mají poslat přímo adresátovi, u něhož však odesílatel zná pouze IP adresu. Pro odeslání prostřednictvím např. Ethernetu ale potřebuje znát cílovou ethernetovou adresu.“ [12]

Z různých experimentů jsem sestavil tato ARP pravidla, podle kterých cisco vyhodnocuje ARP requesty:

1. zdrojová IP adresa není ve stejné síti viz obrázek 5.5, tak se diskartuje ARP request - lze obejít v konfiguraci (jako např. na školních směrovačích)
2. cílová IP adresa nesedí s žádnou s žádnou mojí IP adresou, tak se diskartuje ARP reply
3. IP zdroje (tazatele) je dostupná přes pravidla v routovací tabulce, tak se vygeneruje ARP reply a pošle se tazateli

5.6.5 Pravidla o příjmu paketů

Z předchozí sekce 5.6.4 jsem odvodil několik pravidel pro příjem paketů u cisco směrovače. Tato pravidla tvoří de facto tělo metody `prijmiEthernetove` u `CiscoPocitac`.

- když mohu odpovědět na ARP request a zároveň je paket pro mě nebo vím kam ho poslat dál, tak se paket přijme
- když nelze odpovědět na ARP request = nemám na něj záznam ve směrovací tabulce, tak se paket nepřijme
- ve všech ostatních případech se paket nepřijme

¹⁶Destination Host Unreachable

5.7 Wrapper směrovací tabulky

Routovací tabulka, kterou implementoval kolega, je „šitá“ pro linux. Abych ji mohl použít, musel jsem vytvořit **CiscoWrapper**, který bude linuxovou tabulku ovládat. Hlavní důvodem pro vytvoření nějakého wrapperu je skutečnost, že cisco svoji routovací tabulku vypočítává z nahozených rozhraní a ze statických pravidel vložených uživatelem. Má tedy zvlášť datovou strukturu pro statická pravidla a pro samotnou routovací tabulku.

Mohl jsem si implementovat kompletně vlastní routovací tabulku, ale nějaké podobě wrapperu bych se stejně nevyhnul. Byla by to tedy zbytečná práce, která by navíc znamenala určité zdvojení kódu.

5.7.1 Směrovací tabulka

Linuxová routovací tabulka je složena ze záznamů, kde každý z nich je tvořen těmito položkami: adresát, brána a rozhraní. Existují (pro tuto práci důležité) dva typy záznamů:

- záznam na bránu - příznak UG, při přidávání UG záznamu platí, že nově přidávaná brána musí být dosažitelná příznakem U (= nějakým záznamem na rozhraní)
- záznam na rozhraní - příznak U, převážně záznamy od nahozených rozhraní

5.7.1.1 Výběr záznamů

Pravidla jsou ukládána do routovací tabulky podle masky - ta je hlavním kritériem. Když je tedy potřeba rozhodnout, který záznam vybrat pro příchozí paket, tak se začne procházet tabulka a vrátí se první záznam, kde se shoduje adresát záznamu s adresátem paketu. Tím zajistíme požadavek cisco, aby se směrovalo vždy podle adresáta s nejvyšším počtem jedniček v masce.

5.7.2 Wrapper

CiscoWrapper v podstatě kopíruje datové struktury linuxové routovací tabulky a přidává několik obslužných metod. Největším oříškem byla správná aktualizace routovací tabulky na základě wrapperu. Wrapper si pamatuje statické směrovací záznamy a dle nahozených rozhraní generuje správné záznamy do routovací tabulky.

5.7.2.1 Statické záznamy

Statické směrovací záznamy se zadávají v privilegovaném módu přes příkaz `ip route <adresa> <maska> <brána OR rozhraní>`. Záznam se nepřidá pouze v případě neexistujícího rozhraní a adresy ze zakázaného rozsahu. Do zakázaného rozsahu patří IP adresy ze třídy D a E. Třídy adres jsou popsány v následujícím odstavci. V dohledné době se začnou uvolňovat IP adresy ze třídy E kvůli nedostatku IPv4 adres. Až se tak stane, tak Cisco bude muset vydat aktualizaci svého IOSu, protože v něm momentálně nelze přiřazovat adresy z tohoto rozsahu.

Třída	Začátek 1. bajt	Maska	Síť	Stanic v každé síti
A	0-127	255.0.0.0	126	16 777 214
B	128-191	255.255.0.0	16384	65534
C	192-223	255.255.255.0	2 097 152	254
D	224-239	multicast		
E	240-255	vyhrazeno jako rezerva		

Upravený výpis s Wikipedie [15] a ověřený z webu organizace IANA [18].

Když se přidává záznam s nedosažitelnou bránou, tak IOS nevypíše žádnou chybovou hlášku (na rozdíl od linux, který vypíše `SIOCADDRT: No such process`). Záznam se uloží do wrapperu a je přidán do routovací tabulky ve chvíli, kdy bude dosažitelný.

Při jakékoliv změně IP adresy na rozhraní či změně statických záznamů se smaže celá routovací tabulka a spustí se aktualizací funkce `update`. Ta nejdříve přidá záznamy na rozhraní (dle nastavených adres všech rozhraní¹⁷) a pak začne postupně propočítávat jednotlivé záznamy z wrapperu. Záznam na rozhraní je přidán automaticky pokud výstupní rozhraní není shozené. Záznam na bránu se hledá přes rekurzivní metodu `najdiRozhraniProBranu`.

V mé implementaci je zabudována ochrana proti smyčkám u záznamů na bránu, která limituje délku takového řetězu na 100 záznamů na bránu.

Statická pravidla lze vypsat v privilegovaném módu příkazem `show running-config` a generovaná pravidla (tedy obsah routovací tabulky) přes příkaz `show ip route`.

5.7.3 Vlastnosti

Při testování školního cisco jsem narazil na zajímavé vlastnosti Cisco IOS. Přidával jsem postupně různá statická pravidla a nechal si vypisovat stav routovací tabulky. Nejdříve jsem vložil tyto záznamy v konfiguračním módu:

```
ip route 0.0.0.0 0.0.0.0 FastEthernet0/0
ip route 3.3.3.0 255.255.255.0 2.2.2.2
ip route 8.0.0.0 255.0.0.0 9.9.9.254
ip route 13.0.0.0 255.0.0.0 6.6.6.6
ip route 18.18.18.0 255.255.255.0 51.51.51.9
ip route 51.51.51.0 255.255.255.0 21.21.21.244
ip route 172.18.1.0 255.255.255.252 FastEthernet0/0
ip route 192.168.9.0 255.255.255.0 2.2.2.2
```

Výpis routovací tabulky přes příkaz `show ip route`:

```
51.0.0.0/24 is subnetted, 1 subnets
S 51.51.51.0 [1/0] via 21.21.21.244
18.0.0.0/24 is subnetted, 1 subnets
```

¹⁷V kapitole 5.6 jsem takovému chování říkal „samo-nastavující záznamy.“

```

S~18.18.18.0 [1/0] via 51.51.51.9
    3.0.0.0/24 is subnetted, 1 subnets
S~3.3.3.0 [1/0] via 2.2.2.2
    21.0.0.0/24 is subnetted, 1 subnets
C    21.21.21.0 is directly connected, FastEthernet0/0
S~192.168.9.0/24 [1/0] via 2.2.2.2
    172.18.0.0/30 is subnetted, 1 subnets
S~172.18.1.0 is directly connected, FastEthernet0/0
S~8.0.0.0/8 [1/0] via 9.9.9.254
S~13.0.0.0/8 [1/0] via 6.6.6.6
    192.168.2.0/30 is subnetted, 1 subnets
C    192.168.2.8 is directly connected, FastEthernet0/1
S*   0.0.0.0/0 is directly connected, FastEthernet0/0

```

Potom jsem smazal defaultní záznam 0.0.0.0 0.0.0.0 FastEthernet0/0 a znovu jsem pořídil výpis:

```

    51.0.0.0/24 is subnetted, 1 subnets
S~51.51.51.0 [1/0] via 21.21.21.244
    18.0.0.0/24 is subnetted, 1 subnets
S~18.18.18.0 [1/0] via 51.51.51.9
    3.0.0.0/24 is subnetted, 1 subnets
S~3.3.3.0 [1/0] via 2.2.2.2
    21.0.0.0/24 is subnetted, 1 subnets
C    21.21.21.0 is directly connected, FastEthernet0/0
S~192.168.9.0/24 [1/0] via 2.2.2.2
    172.18.0.0/30 is subnetted, 1 subnets
S~172.18.1.0 is directly connected, FastEthernet0/0
S~8.0.0.0/8 [1/0] via 9.9.9.254
S~13.0.0.0/8 [1/0] via 6.6.6.6
    192.168.2.0/30 is subnetted, 1 subnets
C    192.168.2.8 is directly connected, FastEthernet0/1

```

Jak je vidět, tak se defaultní záznam opravdu smazal (záznam S* opravdu chybí). Po opětovném spuštění příkazu `show ip route` po cca 20 vteřinách vypadá výpis následovně:

```

    51.0.0.0/24 is subnetted, 1 subnets
S~51.51.51.0 [1/0] via 21.21.21.244
    18.0.0.0/24 is subnetted, 1 subnets
S~18.18.18.0 [1/0] via 51.51.51.9
    21.0.0.0/24 is subnetted, 1 subnets
C    21.21.21.0 is directly connected, FastEthernet0/0
    172.18.0.0/30 is subnetted, 1 subnets
S~172.18.1.0 is directly connected, FastEthernet0/0
    192.168.2.0/30 is subnetted, 1 subnets
C    192.168.2.8 is directly connected, FastEthernet0/1

```


Obsah routovací tabulky se dramaticky změnil. Dlouho jsem si lámal hlavu čím to je způsobeno. Ptal jsem se spolužáků co mají CCNA¹⁸ certifikáty a nikdo mi neuměl vysvětlit, jak je možné, že se obsah routovací tabulky může měnit v čase bez nějaké třetí osoby. Dokonce jsem u známého pouštěl `Cisco Packet Tracer` a zkoušel jsem stejné příkazy, ale nebyl jsem schopen to přes tento simulátor zreprodukovat.

Nakonec jsem přišel na to, že školní cisco je natolik pomalé, že není schopno propočítat routovací tabulku v reálném čase, a tak aktualizuje tabulku s několika vteřinovými prodlevami. Prodleva se pohybovala v závislosti na počtu záznamů v rozmezí 5-40 vteřin, zkoušel jsem ale přidat maximálně asi 15 záznamů, protože pak už je výpis routovací tabulky začíná být nepříjemně nepřehledný. Při vyšším počtu záznamů by se prodleva zřejmě zvyšovala.

5.7.4 Odchytky

Při implementaci wrapperu jsem se několikrát odchytil od skutečného cisca:

1. Prodlevu v aktualizaci routovací tabulky jsem neimplementoval, protože studenti na ni nemohou téměř narazit. A když si náhodou student všimne, že obsah tabulky neseďí, tak příkaz pro výpis zpravidla spustí znova a vše bude už v pořádku. Navíc jiná cisca mohou být mnohem rychlejší než ty školní a když takovou vlastnost nemá ani oficiální simulátor, tak asi nemá smysl to implementovat zde.
2. Stanovil jsem limit 100 statických pravidel na bránu propojených do jednoho dlouhého řetězu. Nepřepokládám, že by bylo něco takového potřeba, 100 záznamů by ale mělo být víc než dost. Příklad velmi krátkého řetězu o třech záznamech:

```
ip route 4.4.4.0 255.255.255.0 6.6.6.6
ip route 6.6.6.0 255.255.255.0 8.8.8.8
ip route 8.8.8.0 255.255.255.0 9.9.9.9
..
```

¹⁸Cisco Certified Network Associate

5.8 Překlad adres

Překlad síťových adres neboli NAT (Network address translation) je způsob úpravy paketů síťového provozu, kde se přepisuje zdrojová a/nebo cílová adresa a někdy také i port. Obvykle se také přepočítává kontrolní součet, který by po úpravě adres nebo portu neodpovídal a byl by takový paket na prvním routeru zahozen.

NAT se používá především pro připojení počítačů na lokální síti do internetu, kde jsou takové počítače skryty zpravidla za jednu IP adresu. Navíc technologie překladu adres nepatrně zvyšuje bezpečnost počítačů (připojených za NATem), protože útočník nezná opravdovou IP adresu „zanatovaného“ počítače. Nelze však NAT používat místo zabezpečení.

5.8.1 Cisco a NAT

Cisco podporuje několik druhů překladu [3] síťových adres:

- statický NAT
- dynamický NAT
- overloading
- jakákoliv kombinace statického a dynamického NATu s overloading.

Překlad adres lze u cisco nastavit opravdu detailně, pro tuto práci je však důležitá pouze úzká podmnožina NATu. Detailní popis fungování překladu adres lze nalézt na webových stránkách společnosti Cisco Systems [5]. Soukromý počítač je počítač umístěný v lokální síti. Je tedy schován za nějakou IP adresu routeru, který provádí překlad. Naopak veřejný počítač je takový počítač, který není schován za NATem (alespoň z pohledu soukromého počítače¹⁹).

5.8.1.1 Statický NAT

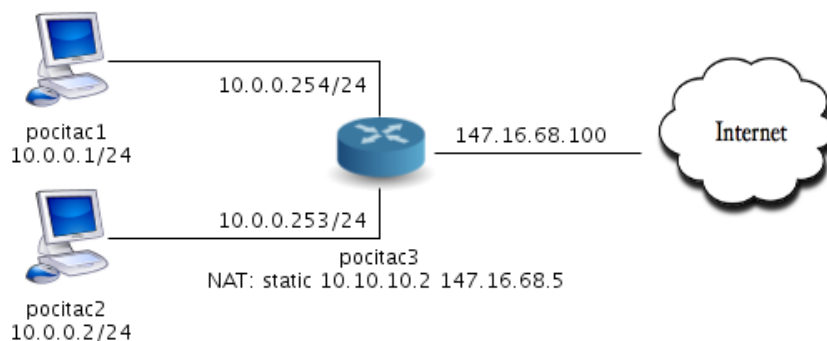
Statický překlad adres umožňuje stanovit pravidla, která říkají, na jakou adresu má být přeložen počítač s danou IP adresou. Např.:

```
ip nat inside source static 10.10.10.2 147.16.68.5
```

Tímto příkazem se bude překládat paket s adresou 10.10.10.2 na adresu 147.16.68.5 (za předpokladu, že paket přišel z rozhraní, které je nastavené jako soukromé, a paket míří do veřejné sítě - přes veřejné rozhraní). Statický NAT funguje i obráceným směrem. Je tedy možné zviditelnit počítač za NATem přes statické pravidlo. Pakety od jiných počítačů se překládat nebudou.

Celá situace je znázorněna na obrázku 5.7. U paketů od `pocitac1` se bude přepisovat zdrojová IP adresa na 147.16.68.5. Zatímco pakety od `pocitac2` zůstanou nezměněny.

¹⁹Ale i takovýto počítač může být schován za jiným nadřazeným routerem za NAT.



Obrázek 5.7: Statický překlad adres

5.8.1.2 Dynamický NAT a overloading

Dynamický překlad adres je velmi podobný statickému s tím rozdílem, že se pravidla generují dynamicky. Je pouze přiřazen pool²⁰ IP adres, ze kterého se přiřazují adresy (s portem) při překladu. Navíc lze omezit adresy sítě, které se mají překládat (příkaz `access-list`).

Overloading je spíše podmnnožina dynamického NATu, kde je povoleno přiřazovat jednu IP adresu více počítačům. Pakety od různých počítačů jsou pak odlišeny jiným portem.

Dynamický překlad probíhá tak, že se nejdříve zkontrolují `access-listy`, zda se má vůbec překládat. Když IP adresa spadá do `access-listu`, tak se vezme volná IP adresa (při metodě overloading může být vybrána jedna adresa vícekrát) z poolu, který je k `access-listu` přiřazen. Touto vybranou IP adresou počítač přepíše zdrojovou adresu příchozího paketu a vytvoří dynamický záznam do NAT tabulky. Zpětný překlad je podstatně jednodušší. Počítač vybere dynamický záznam, přepíše cílovou adresu a předá paket směrovacím pravidlům.

IOS příkazy

Přidat pool adres lze příkazem:

```
ip nat pool <POOL_JMENO> <IP_START> <IP_KONEC> prefix <PREFIX>
```

POOL_JMENO - název poolu IP adres

IP_START - adresa, od které se budou adresy generovat

IP_KONEC - adresa, do které se budou adresy generovat

PREFIX - maska poolu v počtu jedničkových bitů

Přístupový list pro omezení překladu adres se přidá přes příkaz:

```
access-list <CISLO> permit <IP> <WILDCARD>
```

CISLO - identifikátor `access-listu`

IP - adresa sítě povolených IP adres

WILDCARD - maska ve tvaru wildcard (maska = broadcast - wildcard)

²⁰pool je doslova zásoba či rezerva

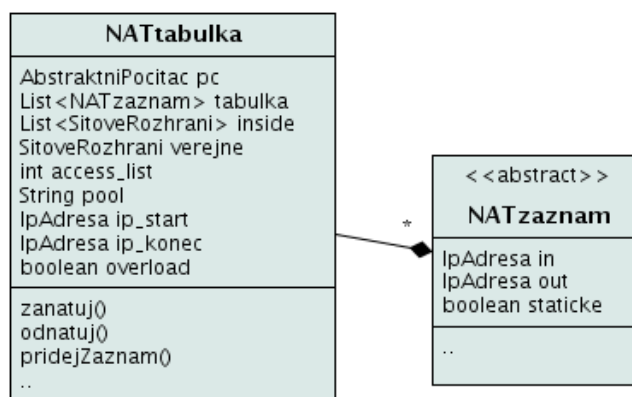
Přiřazení poolu k access-listu se provádí příkazem:

```
ip nat inside source list <ACCESS-LIST> pool <POOL> overload?
ACCESS-LIST - identifikátor access-listu
POOL - jméno poolu
overload - nepovinná volba pro overloading
```

5.8.2 Návrh a implementace NATu

5.8.2.1 Návrh

Každý počítač bude mít vlastní NAT tabulku. Ta bude složená ze seznamu **NATzaznam**. Dále NAT tabulka bude potřebovat seznam soukromých rozhraní a odkaz na jedno veřejné. Bude také potřeba si držet číslo aktuálního access-listu, poolu a stav příznaku overloading. Vše ilustruje obrázek 5.8.



Obrázek 5.8: Návrh překladač adres č.1

5.8.2.2 Implementace

Při implementaci NAT tabulky vyplula na povrch chyba v návrhu. Návrh počítal s tím, že může být pouze jeden pool a jeden access-list. Skutečné cisco si ale pamatuje bez problému i stovky těchto příkazů. Z tohoto důvodu jsem musel původní návrh přizpůsobit. Přidal jsem několik dalších datových struktur, které mi v pozdější fázi velmi usnadnili manipulaci s NAT tabulkou. Na obrázku 5.9 je znázorněn nový návrh.

Postup překladač adres

Podrobný postup překladač adres na skutečném cisco lze nalézt na webových stránkách firmy Cisco [6].

Směr ven

Když přijde paket do počítače, tak se nejdříve provede reverzní překlad²¹. Po té počítač dle routovací tabulky rozhodne, na které rozhraní paket odešle. Pak přichází na řadu samotný akt překladu adres. Chování NATu je řízeno několika pravidly:

- Když příchozí rozhraní²² není označeno jako soukromé²³ nebo odchozí rozhraní není označeno jako veřejné²⁴, tak se paket přepoše dál bez překladu adres.
- Pokud lze nalézt statické pravidlo, které by odpovídalo zdrojové adrese, tak se přeloží zdrojová adresa dle tohoto pravidla.
- Když není nalezen žádný access-list, který by odpovídal zdrojové adrese, tak se paket přepoše bez překladu.
- Když zdrojová adresa paketu spadá do access-listu, ke kterému není přiřazen žádný pool, tak se pošle zpět odesílateli **Destination Host Unreachable**.
- Když v IP poolu, který je přiřazen k odpovídajícímu access-listu, dojdou volné IP adresy, tak se pošle zpět odesílateli **Destination Host Unreachable**.
- V ostatních případech se provede překlad adres:
 1. Zkusí se nalézt odpovídající dynamické pravidlo. V případě nenalezení viz následující bod.
 2. Vygeneruje se nová IP adresa z poolu a vloží se takto vytvořený záznam do NAT tabulky.

Vše je lépe znázorněno vývojovým diagramem 5.10.

Směr dovnitř - reverzní překlad

Reverzní překlad je mnohem jednodušší záležitostí. Příchozí paket čekají tyto pravidla:

1. Nejdříve se zkontroluje, zda paket přišel přes veřejné rozhraní. Když je vstupní rozhraní jiné než veřejné, tak se reverzní překlad neprovede.
2. Pak se prohledají statická pravidla, zda nějaké odpovídá (včetně portu) cílové adrese paketu. Pokud odpovídá, tak se provede reverzní překlad.
3. Když se nenajde ani žádný dynamický záznam, tak se cílová adresa paketu překládat nebude a paket zůstane beze změny.

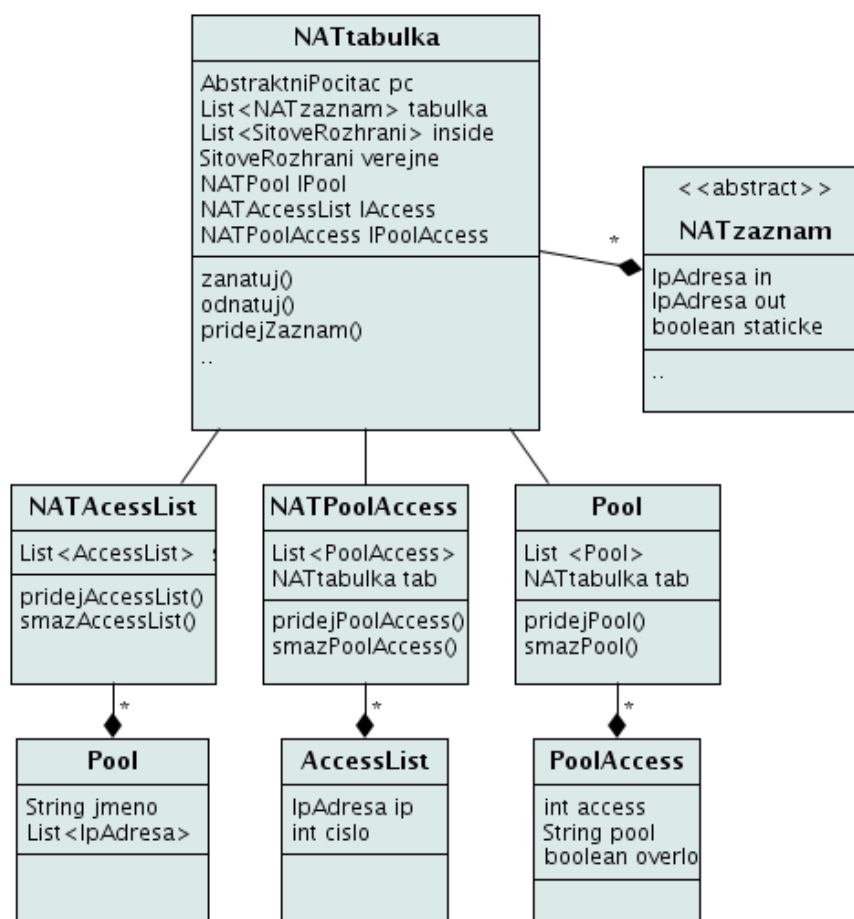
²¹ Reverzní překlad by v češtině nejlépe vystihoval výraz „odnatování“.

²² Rozhraní, pomocí kterého byl paket do počítače přijat.

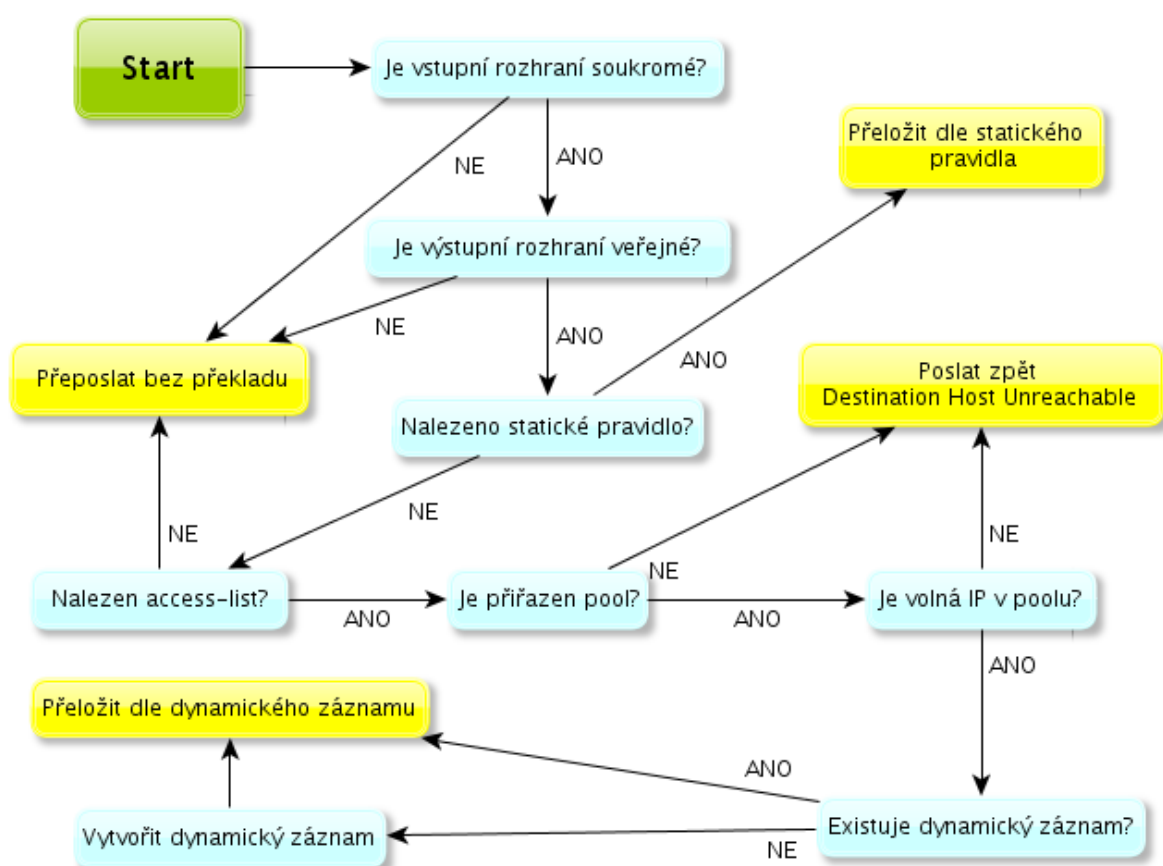
²³ V Cisco IOSu pomocí příkazu „ip nat inside“.

²⁴ Veřejné rozhraní je označeno příkazem „ip nat outside“.

Z obou směrů překladu vyplývá, že statická pravidla mají přednost před dynamickými záznamy. Při testování NATu na školních ciscách jsem zjistil, že cisca zapomínají všechny dynamické záznamy starší cca 10 vteřin. Tato vlastnost byla dodatečně implementována i do této aplikace. Více informací o konfiguraci statického a dynamického NATu zároveň je na stránkách Cisco Systems [8].



Obrázek 5.9: Návrh překladu adres č.2 - finální



Obrázek 5.10: Vývojový diagram překladu adres

Kapitola 6

Testování

6.1 Uživatelské testování

Uživatelské testování bylo prováděno za přítomnosti kolegy z projektového týmu a uživatele - testera, který byl ochoten věnovat dvě hodiny svého času testování. Popis testování linuxové části byl zde vypuštěn.

6.1.1 Zadání

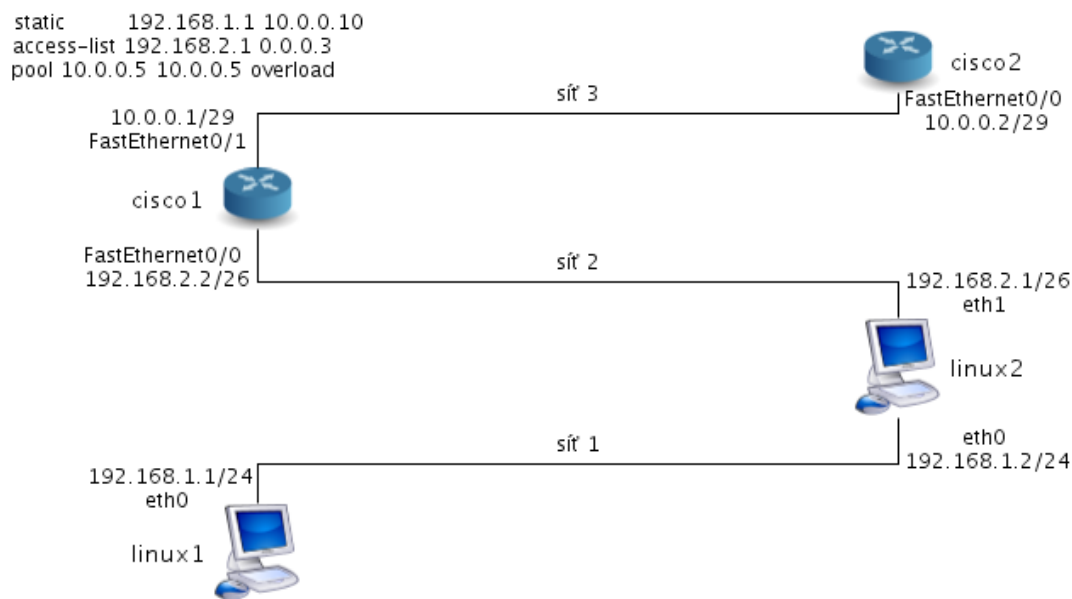
Uživatel měl za úkol otestovat aplikaci v několika krocích:

1. spuštění serveru a připojení klientů pomocí skriptů, uživatel měl k dispozici textový soubor `readme.txt` s informacemi o spuštění
2. kontrola průchodů všech módů Cisco IOS + testování nestandardních vstupů
3. konfigurace rozhraní
4. konfigurace statického překladu adres na počítači `cisco1` pro `linux1`
5. konfigurace dynamického překladu adres na počítači `cisco1` pro `linux2`

Na obrázku [6.1](#) je znázorněno zadání, které dostal uživatel. Uživatel postupoval dle zadaných úkolů a pomocí tohoto obrázku konfiguroval síť.

6.1.2 Průběh testování

U každé objevené chyby bude vyznačeno tučným písmem implementované řešení problému či alespoň komentář s vysvětlením.



Obrázek 6.1: Zadání sítě pro uživatelské testování

6.1.2.1 Úkol č.1 spuštění serveru

Uživatel byl znalý linuxu, a tak pro spuštění serveru zkusil příkaz:

```
./start_server.sh --help
```

Skript však reaguje pouze na spuštění bez parametru a na parametr **-h**. Tudíž se spustil server s konfiguračním souborem **--help** a vypsál chybu, že nelze nic načíst z takového konfiguračního souboru.

(volba `--help` přidána)

Uživatel zkusil spustit server na obsazeném portu. Aplikace vypsala nepříliš jasnou chybovou hlášku.

(chybová hláška upravena)

Klientský skript pro připojení k serveru nevypsál chybovou hlášku, když nebyl specifikován port (pro virtuální počítač).

(chybová hláška přidána)

6.1.2.2 Úkol č.2 průchod stavu IOS

Server vypisuje různé důležité (průchod paketu - směrování, překlad adres) informace. Server ale také vypisuje všechny přijaté a odeslané příkazy, které činí výpis velmi nepřehledným.

(nedůležité výpisy odstraněny)

Uživatel zaznamenal, že při procházení historií příkazů (pomocí šipek nahoru a dolů) se nabízejí příkazy i z jiných terminálů - linux i cisco najednou. Navíc si pamatuje příkazy i po znovu spuštění klienta.

(daň za použití programu třetí strany)

Pomocný příkaz `help` reps. `help_en` není nikde v `readme.txt` zveřejněn, takže uživatel neznal podporované příkazy.

(informace o příkazu `help` přidána do `readme.txt`)

Uživatel našel bug¹ při přechodu z privilegovaného do konfiguračního stavu. Při `configure memory` se přeplo cisco do `configure terminal`.

(chyba opravena)

6.1.2.3 Úkol č.3 nastavení rozhraní

Při konfiguraci síťových rozhraních se přišlo na jednu závažnou chybu. Při nastavování IP adresy zadal uživatel neplatnou masku a program vyhodil výjimku.

(zpracování chybných adres bylo při testování hotové, ale v implementaci chyběl příkaz `return` pro vyskočení z metody při odchycení výjimky; opraveno)

6.1.2.4 Úkol č.4 směrování

Konfigurace směrovacích záznamů odhalila jednu chybu či spíše „nedodělek“. Spuštěný příkaz `ping` bez parametrů nedělá vůbec nic (žádná chybová hláška ani správná funkcionality). **(ping bez parametrů vypíše vysvětlující informaci)**

6.1.2.5 Úkol č.5 statický NAT

Do zadání statického NATu byla předem vložena chyba, aby se overilo, zda se aplikace bude správně chovat. Adresa `192.168.1.1` se měla přeložit na `10.0.0.10`, ta ale nespadá do sítě č.3, a tak `cisco2` nevědělo, kam má paket s odpovědí poslat.

Po změně statického pravidla na záznam `192.168.1.1` na `10.0.0.6` (poslední volná IP adresa sítě č.3) už pakety bez problému došly.

6.1.2.6 Úkol č.6 dynamický NAT

Uživatel zkoušel postupně přidávat pravidla pro `access-list` a pool. Spuštěný příkaz `access-list` bez parametrů měl vypsát hlášku `Incomplete Command.`, ale nic nevypsal. **(přidán výpis do parseru příkazu)**

Po nakonfigurování dynamického překladu adres zkoušel uživatel dostupnost jednotlivých počítačů přes příkaz `ping`. Uživatel přidal do linuxových počítačů defaultní směrovací záznamy na `cisco2`. Do routovací tabulky na `cisco2` se vůbec nezasahovalo (vyjma automaticky nastaveného záznamu na rozhraní). Příkazem `ping` byla ověřena dostupnost `cisco2` z obou linuxových počítačů. Naopak `ping 10.0.0.5` z `cisco2` nezafungoval, protože `linux1` je skryt za NATem a paket by musel být odeslán se správným portem z NAT tabulky. Cisco IOS u příkazu `ping` neumožňuje měnit číslo portu u odesílaného paketu.

¹chyba v programu

6.1.3 Shrnutí

Z počtu chyb lze jasně vyvodit, že uživatelské testování určitě smysl má. Většinu objevených chyb tvoří různé chyby či odchylky v parserech.

6.2 Zátěžové testy

6.2.1 Návrh

6.2.2 Výsledky

Kapitola 7

Možná vylepšení

V tomto projektu je možné pokračovat mnohými vylepšeními, zde jsou některé z nich:

1. Prvním by mohlo být grafické rozhraní pro tvorbu konfiguračního souboru (struktury počítačové sítě), kde by bylo možné přidávat počítače a propojení mezi nimi.
2. Pro lepší ladění problémů při konfiguraci sítě by se jistě hodil i tcpdump. Tento program funguje jako analyzátor monitorující síťový provoz na daném rozhraní.
3. Aplikace podporuje pouze jeden síťový prvek - směrovač, a tak by bylo možné program rozšířit o další prvky např. switch (přepínač) nebo bridge. Přidání těchto prvků by znamenalo plnohodnotnou implementaci 2. linkové vrstvy ISO/OSI modelu.
4. Dalším vylepšením by mohla být možnost propojit virtuální se skutečnou sítí.
5. Aplikace de facto nezpracovává signály (Ctrl+C a Ctrl+Z), pouze je přeposílá operačnímu systému (tuto funkci zajišťuje rlwrap). Pokud bychom se chtěli vracet do privilegovaného módu pomocí Ctrl+Z, tak by bylo nutné implementovat vlastního klienta.

Kapitola 8

Závěr

V této práci se podařilo navrhnout a implementovat simulátor virtuální počítačové sítě cisco. Ve spojení s kolegovu částí je tento program schopen úspěšně odsimulovat virtuální síť složenou ze směrovačů cisco a z počítačů založených na jádru linux. V takové síti je možné otestovat znalosti studentů týkajících se konfigurace rozhraní, statického směrování, dynamického a statického překladu adres. Z výsledků uživatelského testování vyplývá, že provedení takovýchto testů smysl má.

Nepřípravenost studentů na laboratorních cvičení z předmětu Y36PSI Počítačové sítě je v současné době problém. Obyčejní studenti zpravidla nemají přístup ke směrovačům od Cisco Systems, a tak by nasazení této aplikace jako studijní pomůcky mohlo vést ke zvýšení připravenosti těchto studentů¹ na laboratorní cvičení.

¹Za předpokladu, že studenti budou mít motivaci k získání bodů ze cvičení.

Literatura

- [1] Boson Holdings, LLC. Cisco Network Simulator & Router Simulator, 2010. Dostupné z: <<http://www.boson.com/AboutNetSim.html>>.
- [2] Christophe Fillot. Cisco 7200 Simulator, 2010. Dostupné z: <http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator>.
- [3] Cisco Systems, Inc. Configuring Network Address Translation: Getting Started, 2010. Dostupné z: <http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.shtml>.
- [4] Cisco Systems, Inc. Understanding the Ping and Traceroute Commands, 2010. Dostupné z: <http://www.cisco.com/en/US/products/sw/iosswrel/ps1831/products_tech_note09186a00800a6057.shtml>.
- [5] Cisco Systems, Inc. How NAT Works, 2010. Dostupné z: <http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094831.shtml>.
- [6] Cisco Systems, Inc. NAT Order of Operation, 2010. Dostupné z: <http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080133ddd.shtml>.
- [7] Cisco Systems, Inc. Cisco Packet Tracer, 2010. Dostupné z: <http://www.ciscopros.info/web/learning/netacad/course_catalog/PacketTracer.html>.
- [8] Cisco Systems, Inc. Configuring Static and Dynamic NAT Simultaneously, 2010. Dostupné z: <http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080093f31.shtml>.
- [9] Katedra informatiky FEI VŠB-TUO. Virtuální laboratoř počítačových sítí, 2010. Dostupné z: <http://www.cs.vsb.cz/vl-wiki/index.php/Virtuální_laboratoř_počítačových_sítí>.
- [10] MICHEK, J. Diplomová práce. In *Emulátor počítačové sítě*, s. 1–78. České vysoké učení technické v Praze, 2008.
- [11] OMNeT++ Community. OMNeT++ Community Site, 2010. Dostupné z: <<http://www.omnetpp.org/>>.
- [12] Příspěvatelé Wikipedie. Address Resolution Protocol, 2010. Dostupné z: <http://cs.wikipedia.org/wiki/Address_Resolution_Protocol>.

- [13] Přispěvatelé Wikipedie. ICMP, 2010. Dostupné z: <<http://cs.wikipedia.org/wiki/ICMP>>.
- [14] Přispěvatelé Wikipedie. Cisco IOS, 2010. Dostupné z: <<http://cs.wikipedia.org/wiki/Soubor:Cisco-router-1.svg>>.
- [15] Přispěvatelé Wikipedie. IP adresa, 2010. Dostupné z: <http://cs.wikipedia.org/wiki/IP_adresa>.
- [16] Přispěvatelé Wikipedie. Telnet, 2010. Dostupné z: <<http://cs.wikipedia.org/wiki/Telnet>>.
- [17] Přispěvatelé Wikipedie. Unified Modeling Language, 2010. Dostupné z: <<http://cs.wikipedia.org/wiki/UML>>.
- [18] The Internet Assigned Numbers Authority. IANA IPv4 Address Space Registry, 2010. Dostupné z: <<http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml>>.
- [19] ZOHO Corp. Simulation Toolkit 7.0, 2010. Dostupné z: <<http://www.webnms.com/simulator/index.html>>.

Příloha A

Seznam použitých zkratek

CLI Command Line Interface

DOM Document Object Model

DTD Document Type Definition

ICMP Internet Control Message Protocol

IOS Internetwork Operating System

NAT Network Address Translation

OS Operating System

Y36PSI předmět Počítačové sítě

XML Extensible Markup Language

SAX Simple API for XML

Příloha B

UML diagramy

Tato příloha není povinná a zřejmě se neobjeví v každé práci. Máte-li ale větší množství podobných diagramů popisujících systém, není nutné všechny umísťovat do hlavního textu, zvláště pokud by to snižovalo jeho čitelnost.

Příloha C

Instalační a uživatelská příručka

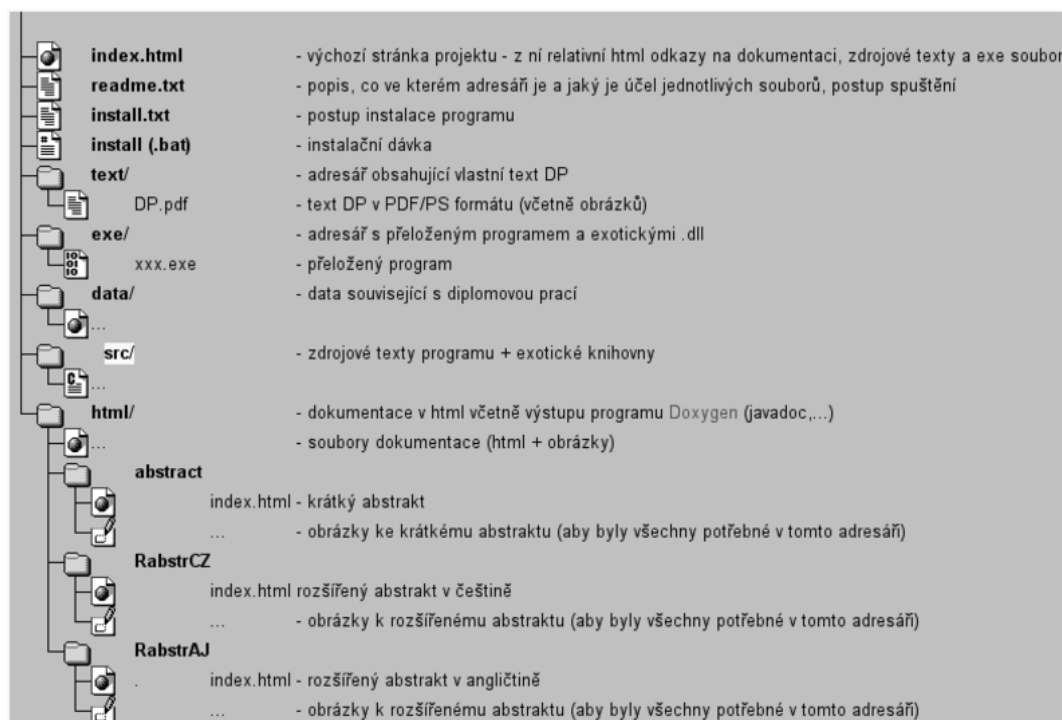
Tato příloha velmi žádoucí zejména u softwarových implementačních prací.

Příloha D

Obsah příloženého CD

Tato příloha je povinná pro každou práci. Každá práce musí totiž obsahovat příložené CD. Viz dále.

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce.



Obrázek D.1: Seznam příloženého CD — příklad

Na GNU/Linuxu si strukturu příloženého CD můžete snadno vyrobit příkazem:

```
$ tree . >tree.txt
```

Ve vzniklém souboru pak stačí pouze doplnit komentáře.

Z **README.TXT** (případně **index.html** apod.) musí být rovněž zřejmé, jak programy instalovat, spouštět a jaké požadavky mají tyto programy na hardware.

Adresář **text** musí obsahovat soubor s vlastním textem práce v PDF nebo PS formátu, který bude později použit pro prezentaci diplomové práce na WWW.