

# 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

20. května 2010



## Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.



## 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 15.5.2010

.....





# Abstract

Translation of Czech abstract into English.

# Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její podstatu. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.

Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.



# Obsah

<b>1</b>	<b>Úvod</b>	<b>1</b>
1.1	Struktura práce . . . . .	1
<b>2</b>	<b>Popis problému, specifikace cíle</b>	<b>3</b>
2.1	Shrnutí funkčních požadavků . . . . .	3
2.2	Nefunkční požadavky . . . . .	4
2.3	Vymezení práce . . . . .	4
<b>3</b>	<b>Existující implementace</b>	<b>5</b>
<b>4</b>	<b>Analýza a návrh řešení</b>	<b>7</b>
4.1	Architektura . . . . .	7
4.1.1	Počátek prací . . . . .	7
4.1.2	Klient - server . . . . .	7
4.1.3	Vyhodnocování příkazů . . . . .	9
4.1.4	Datové struktury jádra . . . . .	9
4.1.4.1	Směrovač - CiscoPocitac . . . . .	9
4.1.4.2	Síťové rozhraní . . . . .	10
4.1.4.3	IP adresa . . . . .	10
4.1.5	Telnet . . . . .	11
4.2	Podobnost simulátoru se skutečným Ciscem . . . . .	11
4.3	Programovací jazyk a prostředí . . . . .	11
4.4	Uživatelské rozhraní . . . . .	12
4.5	Skutečné Cisco . . . . .	12
<b>5</b>	<b>Realizace</b>	<b>13</b>
5.1	Parser Cisco . . . . .	14
5.1.1	Cisco IOS . . . . .	14
5.1.1.1	Uživatelský mód . . . . .	14
5.1.1.2	Privilegovaný mód . . . . .	14
5.1.1.3	Konfigurační mód . . . . .	15
5.1.1.4	Konfigurace rozhraní . . . . .	15
5.1.2	Implementace Cisco IOS . . . . .	16
5.1.3	Odchytky v implementaci . . . . .	17
5.2	Načítání ze souboru . . . . .	18
5.2.1	Zpracování parametrů . . . . .	18

5.2.2	Konfigurační soubor	18
5.2.3	Implementace SAX handleru	19
5.2.3.1	Načítání	19
5.2.3.2	Datová struktura	19
5.2.3.3	Vytváření virtuálních počítačů	20
5.3	Ukládání do souboru	20
5.4	Směrování	21
5.4.1	Debuging	21
5.4.2	Síť č.1	21
5.4.2.1	Popis problému	21
5.4.2.2	Řešení	22
5.4.3	Síť č.2	22
5.4.3.1	Popis sítě	22
5.4.3.2	Experimenty	23
5.4.4	ARP protokol	24
5.4.5	Pravidla o příjmu paketů	24
5.5	Wrapper směrovací tabulky	25
5.5.1	Směrovací tabulka	25
5.5.1.1	Výběr záznamů	25
5.5.2	Wrapper	25
5.5.2.1	Statické záznamy	25
5.5.3	Vlastnosti	26
5.5.4	Odchylky	28
5.6	Překlad adres	29
5.6.1	Cisco a NAT	29
5.6.2	Implementace NATu	29
5.6.3	Datové struktury	29
5.6.3.1	Pooly	29
5.6.3.2	Access-listy	29
5.6.3.3	Přiřazení	29
<b>6</b>	<b>Testování</b>	<b>31</b>
6.1	Uživatelské testování	31
6.1.1	Zadání	31
6.1.2	Průběh testování	31
6.1.3	Nalezené chyby	31
6.1.4	Shrnutí	31
6.2	Zátěžové testy	32
6.2.1	Návrh	32
6.2.2	Výsledky	32
<b>7</b>	<b>Závěr</b>	<b>33</b>
7.1	Možnosti vylepšení	33
<b>A</b>	<b>Testování zaplnění stránky a odsazení odstavců</b>	<b>37</b>

<b>B Pokyny a návody k formátování textu práce</b>	<b>41</b>
B.1 Vkládání obrázků . . . . .	41
B.2 Kreslení obrázků . . . . .	42
B.3 Tabulky . . . . .	42
B.4 Odkazy v textu . . . . .	43
B.4.1 Odkazy na literaturu . . . . .	43
B.4.2 Odkazy na obrázky, tabulky a kapitoly . . . . .	45
B.5 Rovnice, centrovaná, číslovaná matematika . . . . .	45
B.6 Kódy programu . . . . .	46
B.7 Další poznámky . . . . .	46
B.7.1 České uvozovky . . . . .	46
<b>C Seznam použitých zkratk</b>	<b>47</b>
<b>D UML diagramy</b>	<b>49</b>
<b>E Instalační a uživatelská příručka</b>	<b>51</b>
<b>F Obsah přiloženého CD</b>	<b>53</b>



# Seznam obrázků

4.1	Počáteční návrh . . . . .	8
4.2	Návrh komunikační části . . . . .	8
4.3	Zjednodušený diagram tříd . . . . .	9
4.4	Uživatelské rozhraní pod OS Linux . . . . .	12
5.1	Třídní model předků . . . . .	13
5.2	Přehled základních módů Cisco IOS [5] . . . . .	15
5.3	Síť linux - cisco . . . . .	22
5.4	Síť linux1 - cisco1 - cisco2 . . . . .	22
B.1	Popiska obrázku . . . . .	42
F.1	Seznam přiloženého CD — příklad . . . . .	53





# Seznam tabulek

B.1 Ukázka tabulky . . . . .	42
------------------------------	----



# Kapitola 1

## Úvod

Úkolem této práce je navrhnout a implementovat aplikaci, která umožní vytvoření virtuální počítačové sítě pro předmět Y36PSI<sup>1</sup>. 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 IOS<sup>2</sup>. 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 „pohraje“ si s virtuálním ciscem, ke kterému běžný smrtelník nemá přístup. Zjistí, jak to funguje a pak už jen přijde na cvičení předmětu a vše nakonfiguruje tak, jak to má být.

Jelikož tento projekt přesahuje rozsah jedné bakalářské práce, tak byly vymezeny hranice, aby se tento úkol mohl rozdělit na dvě části. Nakonec celá aplikace byla rozdělena na části tři. První je část společná, kde je implementováno jádro klient - server. Druhá část je Cisco IOS, tu jsem dostal na starost já<sup>3</sup>. A třetí část je platforma Linux, kterou zpracoval Tomáš Pitřinec v bakalářské práci Simulátor virtuální počítačové sítě Linux.

### 1.1 Struktura práce

Tady bude popis členění práce na jednotlivé sekce.

---

<sup>1</sup>Počítačové sítě

<sup>2</sup>Internetwork Operating System je operační systém používaný na směrovačích a přepínačích firmy Cisco Systems

<sup>3</sup>Oba jsme chtěli programovat linuxovou část, protože s OS Linux máme oba zkušenosti. Po losování „Černý Petr - Cisco“ padlo na mne.



## Kapitola 2

# Popis problému, specifikace cíle

Nejdříve bylo potřeba zjistit přesné požadavky, tedy co všechno má být tímto simulátorem podporováno. Virtuální síť musí podporovat několik k sobě připojených počítačů (Linux nebo Cisco, vymezení práce viz 2.3). Limit připojených počítačů nebyl v zadání určen, nicméně se počítá s tím, že systém zvládne desítky až stovky počítačů (viz Zátěžové testy), ačkoliv v praxi jich většinou nebude potřeba více než deset. Systém umožní nakonfigurování rozhraní potřebných pro propojení sítě včetně příkazu pro zapínání a vypínání rozhraní. Dále aplikace musí umožnit správu směrovací tabulky pomocí příkazů Cisco IOS. V předmětu Y36PSI se také požaduje po studentech, aby rozuměli tzv. „natování“ neboli překlad adres - NAT<sup>1</sup>. Cisco podporuje hned několik druhů překladu adres. Pro tuto aplikaci jsem zvolil 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. 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 implementován příkazy ping a traceroute.
7. K jednotlivým počítačům aplikace se připojuje pomocí telnetu.
8. Pomocí telnet klientů musí být možné se připojit k jednomu počítači paralelně v několika terminálech najednou.

---

<sup>1</sup>Network Address Translation

## 2.2 Nefunkční požadavky

1. Aplikace bude multiplatformní - alespoň pro OS<sup>2</sup> Windows a Linux
2. Aplikace musí být spustitelná na běžném<sup>3</sup> studentském počítači.
3. Systém by měl být co nejvěrnější kopií reálného Cisca v mezích zadání.

## 2.3 Vymezení práce

Jak už jsem zmínil v kapitole 1, práce byla 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 de facto 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 Realizace 5.4). Díky tomu, že Cisco má svůj překlad adres natolik robustní (díky svým rozsáhlým datovým strukturám), tak se nechalo s menšími úpravami (přidáním několika metod) použít pro linuxový příkaz `iptables`.

Mojí prací je také načítání a ukládání do souboru, startovací skripty pro server i klienty. Co jsem si ale nechal nakonec, jsou různé datové struktury pro jádro celého systému. Na těch jsem spolupracoval s kolegou nejvíce, protože musely odrážet situaci na obou systémech. Např. třída zaštiťující IP adresu je z půlky má a z půlky kolegy. Obvykle je u třídy napsáno, kdo je jejím autorem, zde je to přímo rozlišováno v jednotlivých metodách. Pak je v hlavičce třídy poznámka, že jsme se podíleli oba. Jinde je celá datová struktura kolegy a moje jsou pouze 2-3 metody. Dále síťová část je výsledkem společné práce<sup>4</sup> z roku 2008, kdy jsme právě pro předmět Počítačové sítě programovali hru Karel.

---

<sup>2</sup>Operační systém

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

<sup>4</sup>Dle tehdejších pravidel jsme mohli programovací úlohy implementovat a odevzdávat ve dvojicích.

## Kapitola 3

# Existující implementace

Tady budou zpracovány existující implementace. Když to bude moc dlouhé, tak to dám do extra kapitoly.

Cisco packet tracer, OMNeT++ simulator a další.





## Kapitola 4

# Analýza a návrh řešení

Jádro aplikace bylo vytvářeno ve spolupráci s kolegou, tudíž následující řádky týkající se architektury systému se mohou v nějaké podobě objevit i v jeho práci. Práce se úmyslně nezabývá striktně mojí vlastní „Cisco částí“, protože tento systém tvoří jeden celek, který je ovlivněn jeho podsystémy.

V této kapitole je popsáno především společné jádro. Návrh a implementace Cisco části je v kapitole Realizace 5.

### 4.1 Architektura

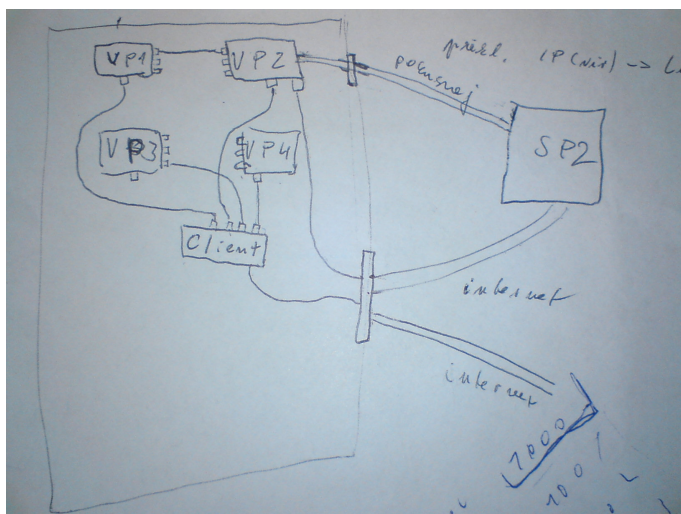
Celá aplikace je rozdělena na dvě vrstvy - síťová a aplikační. Síťová vrstva je více popsána v sekci 4.1.2. Aplikační vrstvu tvoří de facto zbytek systému (směrování, překlad adres, parsery příkazů, ..).

#### 4.1.1 Počátek prací

Když jsem dostal ústní zadání této práce, tak jsem hned začal přemýšlet, jak navrhnout celou aplikaci. Zadání ale nebylo specifikováno přesně se všemi potřebnými detaily, takže jsem měl „volnou“ ruku co se návrhu týče. Po konzultaci s kolegou vzniklo několik variant, na obrázku 4.1 je jedna z nich. Tato varianta počítala s tím, že počítač bude připojen do reálné sítě, ale byla zavržena po dohodě s vedoucím práce kvůli složitosti a náročnosti takového systému.

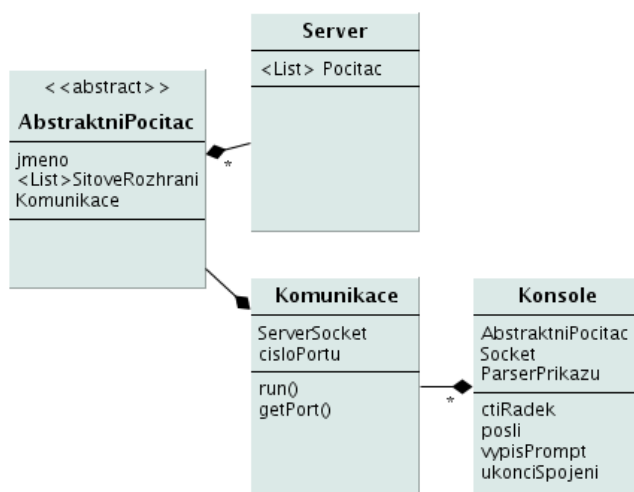
#### 4.1.2 Klient - server

Síťovou vrstvu systému tvoří architektura klient - server. Ta, jak jsem již naznačoval v kapitole 2.3, byla převzata ze semestrální práce, kde bylo za úkol mimo jiné implementovat více-vláknový server. 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



Obrázek 4.1: Počáteční návrh

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 ssh<sup>1</sup> či telnet.



Obrázek 4.2: Návrh komunikační části

Na obrázku 4.2 je znázorněna komunikační část pomocí UML<sup>2</sup> diagramu. Každý počítač má objekt **Komunikace**, která 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.

<sup>1</sup>Secure Shell - zabezpečený komunikační protokol (v současné době náhrada telnetu)

<sup>2</sup>Unified Modeling Language, UML je v softwarovém inženýrství grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.[9]

### 4.1.3 Vyhodnocování příkazů

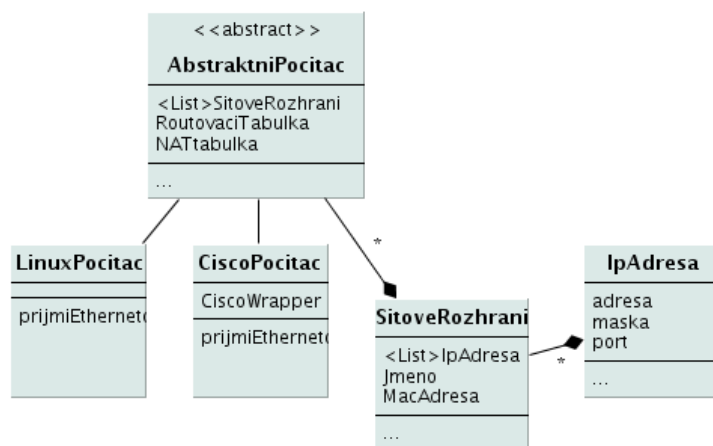
Když uživatel zadá příkaz a ukončí ho znakem nového řádku (klávesa Enter, znak `\n`), tak se ve třídě `Konzole` zavolá metoda `zpracujRadek()`. Tuto metodu vlastní abstraktní `ParserPrikazu`, který je v mém případě implementován jako `CiscoParserPrikazu`<sup>3</sup>. Ten se stará o zpracování poslané řádky a podle toho volá různé Cisco příkazy, které tvoří můj IOS, či jiné servisní příkazy.

### 4.1.4 Datové struktury jádra

Po architektuře klient - server bylo potřeba domyslet datové struktury virtuálních počítačů. Ze začátku jsem nastínil základní třídy a zbytek jsem dodělával jak bylo potřeba.

#### 4.1.4.1 Směrovač - CiscoPocitac

Na skutečné počítačové síti jsou síťové prvky několika druhů (switche, bridge, repeatery, směrovače, ...), ale na laboratorních cvičeních předmětu Y36PSI se nastavují pouze směrovače ze 3. vrstvy<sup>4</sup> síťového ISO/OSI modelu. Proto v mé práci figuruje pouze jeden typ síťového prvku - směrovač (router), který je reprezentován datovou strukturou `CiscoPocitac` viz obrázek 4.3. Ten je spojen přes vlastní rozhraní k jinému počítači přes jeho rozhraní právě jedním „kabelem“. Není tedy možné, aby bylo jedno rozhraní připojeno k více rozhraním, protože jsou to všechno dvou bodové spoje.



Obrázek 4.3: Zjednodušený diagram tříd

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

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

<sup>4</sup>Tato „síťová vrstva“ se stará o směrování v síti a síťové adresování. Dále poskytuje spojení mezi vzdálenými sítěmi, které spolu přímo nesousedí.

obsažena dvakrát. V dnešní verzi programu je to zjednodušené tak, že samotné rozhraní nenese žádnou informaci o kabelu. Ale kabely jsou konfiguračním souboru zvlášť:

```
<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.

#### 4.1.4.2 Síťové rozhraní

Datová struktura pro síťové rozhraní je ve své podstatě jednoduchá. Obsahuje jméno, seznam IP adres přiřazených k tomuto rozhraní, MAC<sup>5</sup> 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 ARP<sup>6</sup> protokol není u nás 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. Tento ukazatel jsem zavedl, protože rozhraní Cisca jsou ve výchozím stavu vypnutá.

#### 4.1.4.3 IP adresa

IP adresa je mnohem složitější 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, co je potřeba s adresou dělat.

<sup>5</sup>MAC - Media Access Control, je fyzická adresa, kterou používá 2. (spojová) vrstva ISO/OSI modelu

<sup>6</sup>„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.“[2]

### 4.1.5 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.[8]“. Podle protokolu se vše posílá po znaku a protistrana po znaku vše potvrzuje. Protokol telnet ale zas tak jednoduchý není. Podporuje několik režimů, při navazování spojení začne proces vyjednávání atd. To všechno implementovat by bylo na samostatnou (možná i diplomovou) práci.

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`. Tato „vlastnost“ 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 BASH<sup>7</sup>. V případě této aplikace toho ale nelze využít, tak jsem se rozhodl, že tyto funkcionality budou na straně klienta, kde to bude zajišťovat „někdo jiný“.

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 je to vyřešeno tak, že se vše pouští pod programem Cygwin. Navíc toto řešení zvyšuje komfort práce pod Windows, jelikož program `cmd` není úplně uživatelsky přívětivý.

## 4.2 Podobnost simulátoru se skutečným Ciscem

Při implementaci jsem se snažil vytvořit systém, který bude co nejvíce podobný skutečnému Cisco. Musel jsem ale někde položit hranici mezi složitostí a věrností výsledné práce, protože tyto dvě metriky jsou vzájemným protikladu. Cisco IOS je natolik robustní a pracovaný systém, že je v mých silách pouze implementace úzké části systému, která je nutně potřeba pro splnění cíle. Byl jsem přinucen místy ustoupit a nechat vypsat hlášení, že to či ono není podporováno. V samotném parseru příkazů není toto téměř vůbec řešeno, protože by to znamenalo dopsání dohromady 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 podporováno a co ne, tak jsem přidal příkaz `help` (`help_en` pro výpis v angličtině), který popisuje, co lze v jakém stavu Cisca použít.

## 4.3 Programovací jazyk a prostředí

Pro implementaci simulátoru jsem si vybral 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 dozvíme víc, než v jazyce C++ s

---

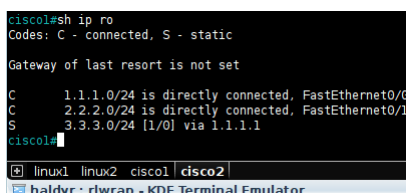
<sup>7</sup>Bourne again shell - nejpoužívanější unixový shell

jeho `Segmentation fault`. Neméně významným důvodem je i skutečnost, že s Javou mám zatím největší zkušenosti.

Celou práci jsem implementoval v Netbeans IDE<sup>8</sup> verze 6.8.

## 4.4 Uživatelské rozhraní

Uživatelské rozhraní je v zásadě velmi jednoduché. Vše je ovládáno přes příkazovou řádku, tak jak jsme zvyklí. Spuštění serveru je ulehčeno pomocným skriptem `start_server.sh`, který zároveň obsahuje nápovědu. Pro připojování klientů jsem připravil skripty, ve kterých je zohledněna verze programu rlwrap. V balíčku programu Cygwin je rlwrap starší verze, která neumožňuje obarvování promptu a přeposílání signálů (např. při zmáčknutí Ctrl+C nebo Ctrl+Z). Novější verze už tyto funkce mají. Skripty pro připojení na `linux.sh` a `cisco.sh` fungují nezávisle na verzi.



```
cisco1#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
cisco1#
```

Obrázek 4.4: Uživatelské rozhraní pod OS Linux

## 4.5 Skutečné Cisco

Jedním z největších „oříšků“ této práce bylo zjistit, jak se chová skutečné Cisco. Pravdou je, že Cisco Systems má na svých stránkách slušnou řadu návodů, nicméně není tak jednoduché v nich najít, co zrovna potřebujeme. A tak jsem hodně věcí zjišťoval z živého Cisca umístěného na Karlově náměstí přes protokol ssh.

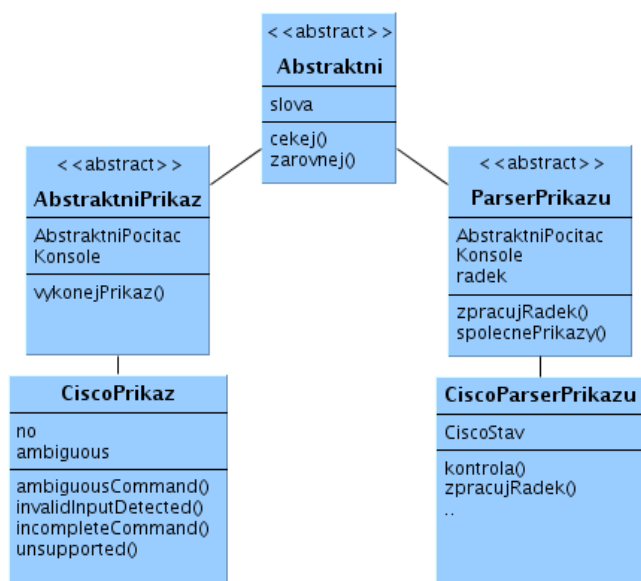
---

<sup>8</sup>Integrated Development Environment

## Kapitola 5

# Realizace

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 speciální metody jen pro Cisco příkazy. Viz obrázek 5.1.



Obrázek 5.1: Třídní model předků

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

## 5.1 Parser Cisco

Po návrhu a implementaci jádra systému se moje úsilí přesunulo k parseru příkazů pro Cisco.

### 5.1.1 Cisco IOS

Cisco IOS je operační systém, který se nachází na drtivě většině směrovačů firmy Cisco Systems. IOS obsahuje pouze ovládání přes příkazový řádek - CLI<sup>1</sup>. Pro mě je to spíše výhodou, protože je to mnohem jednodušší na implementaci ve srovnání s „klikacím“ GUI<sup>2</sup>. IOS má implementováno tzv. zkracování příkazů, které zefektivňuje práci s celým systémem. Celé to funguje tak, že když uživatelův začátek příkazu lze doplnit na 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 5.2 jsou zobrazeny důležité stavy IOS a přechody mezi nimi.

#### 5.1.1.1 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. výpis směrovací tabulky `show ip route` či příkazy `ping` nebo `traceroute`. Do privilegovaného režimu se lze přepnout příkazem `enable`.

#### 5.1.1.2 Privilegovaný mód

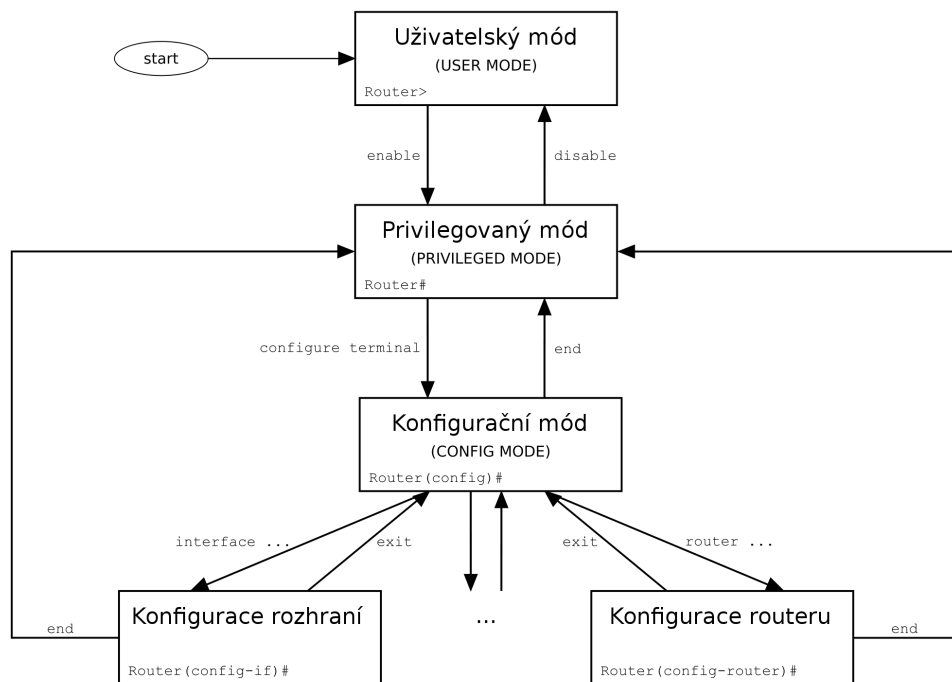
Privilegovaný mód (PRIVILEGED MODE) nebo také „administrátorský“ mód je podobný linuxovému `root` účtu. Tento mód je výchozím bodem pro vstup do ostatních módů. Pro návrat zpět do uživatelského režimu existuje příkaz `disable`. Příkaz `configure` způsobí přepnutí do dalšího konfiguračního módu. Tento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

---

<sup>1</sup>Command Line Interface

<sup>2</sup>Graphical User Interface





Obrázek 5.2: Přehled základních módů Cisco IOS [5]

- `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`.

#### 5.1.1.3 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`).

#### 5.1.1.4 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`.

### 5.1.2 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 Ciscu 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.

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 jak to vypadá v kódu:

```
if (cmd.length() >= i && command.startsWith(cmd)) {
    // lze doplnit na jeden jedinecny prikaz
    return true;
}
if (command.startsWith(cmd)) {
    // vypsát amiguous command
    nepokracovat = true;
}
```

Jednotlivé příkazy Cisco IOS jsou implementovány v samostatných třídách. Třída `CiscoParserPrikazu` tedy zajišťuje přechody mezi stavy (módy) a „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 věcí je vhodné spíše pro ladění programu do budoucna než pro běh v „ostrém“ provozu. Tento mód je ve výchozím stavu vypnut.

### 5.1.3 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 standardní výstup. Je tedy už nepoužitelný a pro tyto přídady existuje příkaz `kill`, který ukončí tento rozbitý proces a klient se může připojit znova. Lépe je na tom 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 verzi<sup>3</sup>, která neumožňuje přeposílání signálů, takže při stisku `Ctrl+C` i `Ctrl+Z` přestane `telnet` klient vypisovat na standardní výstup a nezbývá než použít `kill`.

---

<sup>3</sup>stav z 18.5.2010

## 5.2 Načítání ze souboru

### 5.2.1 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 vůbec neřeší. 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, výchozí volba je port 4000. Když je daný port obsazený, tak se program ukončí z chybovou hláškou.

### 5.2.2 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 KDE<sup>4</sup>, kde jednotlivé sekce jsou odděleny jménem v hranatých závorkách

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

- XML<sup>5</sup> 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.

<sup>4</sup>K Desktop Environment je desktopové prostředí pro Linux a další unixové operační systémy.

<sup>5</sup>Extensible Markup Language je rozšiřitelný značkovací jazyk

### 5.2.3 Implementace SAX handleru

Zpracovat XML lze přes technologii DOM<sup>6</sup> a SAX<sup>7</sup>. Já se rozhodl pro SAX z těchto důvodů:

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

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

#### 5.2.3.1 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. Musí být implementovány 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 DTD<sup>8</sup>, který definuje strukturu XML dokumentu.

#### 5.2.3.2 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 Linuxu
- 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.

---

<sup>6</sup>Document Object Model

<sup>7</sup>Simple API for XML

<sup>8</sup>Document Type Definition

### 5.2.3.3 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) všechny uložené 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), natažená kabeláž 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>
```

## 5.3 Ukládání do souboru

Abstraktní `ParserPrikazu` obsahuje metodu, do které jsou vloženy všechny společné příkazy. 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 a jednoduchost 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 musíme přidat pravidla pro načítání z XML v `SAXHandler` a navíc zde ukládání.

## 5.4 Směrování

Směrování implementoval kolega, nicméně se Cisco 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 to tak je. 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 už chovají jinak.

Při různých experimentech jsem zjistil, že když linuxu přijde paket, na který nemá záznam (routu) 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 síť nakonfigurovat trochu neobvykle, tak to tak jasné nebylo.

### 5.4.1 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 tyto příkazy:

- `debug ip packet detail` - detailní výpis zpracování IP paketů
- `debug ip icmp` - zpracování ICMP<sup>9</sup> paketů
- `debug arp` - výpis ARP protokolu o zjišťování MAC adres sousedních počítačů

### 5.4.2 Síť č.1

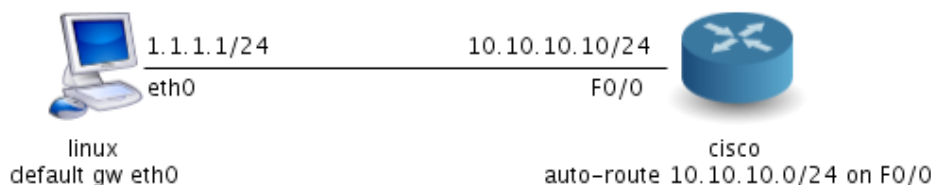
#### 5.4.2.1 Popis problému

Vezměme si následující síť 5.3 složenou pouze ze dvou počítačů cisco a linux. Tyto počítače mají nastavené IP adresy z jiných sítí, linux má nastavenou defaultní routu na rozhraní `eth0` a cisco má samo-nastavující se routu na síť, která je přiřazena na rozhraní `F0/0`<sup>10</sup>. Samo-nastavující znamená, že cisco přidává záznamy do routovací tabulky podle informací z jeho rozhraní. Tuto funkcionalitu lze vypnout, na školních ciscích je ale defaultně zapnutá. Cisco nenahazuje tuto routu v případě, že na druhém konci kabelu buď nikdo není nebo je shozené (vypnuté) rozhraní.

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 **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.

<sup>9</sup>Internet 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. [4]

<sup>10</sup>Na obrázcích sítí se vyskytují zkratky `F0/0` a `F0/1`, které značí FastEthernet0/0 resp. FastEthernet0/1



Obrázek 5.3: Síť linux - cisco

#### 5.4.2.2 Řešení

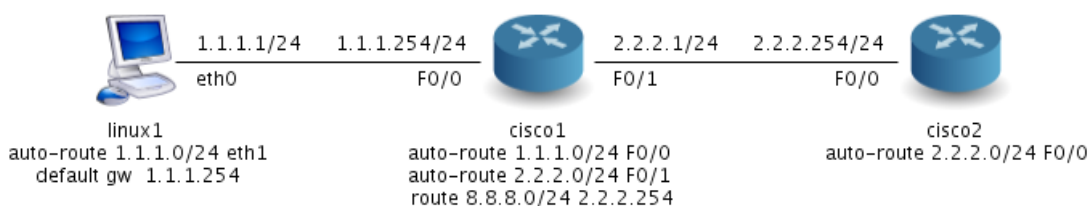
Nejdříve vysvětlím proč prvních několik paketů prošlo až na cisco a na další `icmp_req` odpověděl linux sám sobě. Je to způsobeno tím, že při prvních paketech ještě cisco nevědělo co s těmi pakety bude, a tak je přijalo, aby mohlo vyhodnotit co dál. Cisco ale hned přišlo na to, že nemá žádnou routu na 1.1.1.1, takže neví, co s takovými pakety dělat, tak se radši rozhodlo, že je ani nepřijme. Obvykle cisco nepřijímá pakety ihned, ale školní cisca měly starší verzi software a celkově byly zpomalené, takže to bylo způsobeno asi tím.

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

#### 5.4.3 Síť č.2

##### 5.4.3.1 Popis sítě

Na další síti 5.4 jsou počítače linux1, cisco1 a cisco2 zapojené do jedné „nudle“. Konfigurace rozhraní a směrovacích tabulek je obsažena v obrázku. Linux1 má teoreticky v dosahu (přes defaultní routu) 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 5.4: Síť linux1 - cisco1 - cisco2



### 5.4.3.2 Experimenty

#### I. experiment

První experiment je ping z linux1 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.
From 2.2.2.254 icmp_seq=1 Destination Host Unreachable
From 2.2.2.254 icmp_seq=2 Destination Host Unreachable
```

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 poslalo zpátky odesílateli 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. Je to zapříčiněno také tím, že routa 1.1.1.0/24 je na rozhraní a ne na gateway.

#### 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 vypsalo „“, což znamená, že vypršel timeout. Linux poslal DHU<sup>11</sup> a ciscu vypršel timeout.

---

<sup>11</sup>Destination Host Unreachable

#### 5.4.4 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.“[2]

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

1. zdrojová IP adresa není ve stejné síti viz obrázek 5.3, tak se diskartuje ARP request - lze obejít v konfiguraci, tak jsou také nastavená školní cisca
2. cílová IP adresa nesedí s žádnou s žádnou mojí IP adresou, diskartuje se ARP reply
3. IP zdroje (tazatele) je dostupná před pravidla v routovací tabulce, tak se vygeneruje ARP reply a pošle se tazateli

#### 5.4.5 Pravidla o příjmu paketů

Z předchozí sekce 5.4.4 jsem vyvodil několik pravidel pro příjem paketů u počítače cisco. 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 routu ve směrovací tabulce, tak se paket nepřijme
- ve všech ostatních případech se paket nepřijme

## 5.5 Wrapper směrovací tabulky

Routovací tabulka, kterou implementoval kolega, je „šitá“ pro linux. Abych ji mohl použít, musel jsem vytvořit **CiscoWrapper**, který bude v podstatě 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.

### 5.5.1 Směrovací tabulka

Linuxová routovací tabulka je složena ze záznamů, každý z nich je tvořen těmito položkami: adresát, brána a rozhraní. Existují 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

#### 5.5.1.1 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.

### 5.5.2 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.

#### 5.5.2.1 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	0-127	255.0.0.0	126	16 777 214
B	10	128-191	255.255.0.0	16384	65534
C	110	192-223	255.255.255.0	2 097 152	254
D	1110	224-239	multicast		
E	1111	240-255	vyhrazeno jako rezerva		

Upravený výpis s Wikipedie [6] a z webu organizace IANA[11].

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 routy na rozhraní (dle nastavených adres všech rozhraní<sup>12</sup>) 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`.

### 5.5.3 Vlastnosti

Při testování školního cisca jsem narazil na zajímavé vlastnosti Cisco IOSu. 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
S      18.18.18.0 [1/0] via 51.51.51.9
```

<sup>12</sup>V kapitole 5.4 jsem takovému chování říkal „samo-nastavující routy.“

```

    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 routu 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 routa opravdu smazala (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 CCNA<sup>13</sup> 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 navyšovala.

#### 5.5.4 Odchytky

Při implementaci wrapperu jsem se několikrát odchytil od skutečného cisco:

1. Prodlevu v aktualizaci routovací tabulky jsem neimplementoval, protože studenti na ní nemohou téměř narazit. A když si náhodou student všimne, že obsah tabulky nesedí, tak příkaz pro výpis zpravidla spustí znova a vše bude už v pořádku. Navíc jiná cisco 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:

```
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
..
```

---

<sup>13</sup>Cisco Certified Network Associate

## 5.6 Překlad adres

Tady bude popsán překlad adres. Co to je, k čemu slouží ..

### 5.6.1 Cisco a NAT

Druhy NATu u cisco

### 5.6.2 Implementace NATu

#### 5.6.2.1 Datové struktury





## Kapitola 6

# Testování

### 6.1 Uživatelské testování

#### 6.1.1 Zadání

#### 6.1.2 Průběh testování

#### 6.1.3 Nalezené chyby

#### 6.1.4 Shrnutí

## 6.2 Zátěžové testy

### 6.2.1 Návrh

### 6.2.2 Výsledky

# Kapitola 7

## Závěr

### 7.1 Možnosti vylepšení

- Zhodnocení splnění cílů DP/BP a vlastního přínosu práce (při formulaci je třeba vzít v potaz zadání práce).
- Diskuse dalšího možného pokračování práce.
  - graficke klikatko pro tvorbu XML konfiguraku
  - tcpdump
  - switche
  - napojeni na realnou sit
  - zpracovani signalu Ctrl+C, Ctrl+Z (-> vlastni klient, kouzlo telnetu je pak pryc)



# Literatura

- [1] HAINDL, M. – KMENT, Ľ. – SLAVÍK, P. Virtual Information Systems. In *WSCG'2000 — Short communication papers*, s. 22–27. University of West Bohemia, Pilsen, 2000.
- [2] Příspěvatelé Wikipedie. *Address Resolution Protocol* [online]. 2010. [cit. 14.5.2010]. Dostupné z: <[http://cs.wikipedia.org/wiki/Address\\_Resolution\\_Protocol](http://cs.wikipedia.org/wiki/Address_Resolution_Protocol)>.
- [3] Příspěvatelé Wikipedie. *Framework* [online]. 2009. [cit. 10.9.2009]. Dostupné z: <<http://cs.wikipedia.org/wiki/Framework>>.
- [4] Příspěvatelé Wikipedie. *ICMP* [online]. 2010. [cit. 19.5.2010]. Dostupné z: <<http://cs.wikipedia.org/wiki/ICMP>>.
- [5] Příspěvatelé Wikipedie. *Cisco IOS* [online]. 2010. [cit. 16.5.2010]. Dostupné z: <<http://upload.wikimedia.org/wikipedia/commons/thumb/0/04/Cisco-router-1.svg/600px-Cisco-router-1.svg.png>>.
- [6] Příspěvatelé Wikipedie. *IP adresa* [online]. 2010. [cit. 18.5.2010]. Dostupné z: <[http://cs.wikipedia.org/wiki/IP\\_adresa](http://cs.wikipedia.org/wiki/IP_adresa)>.
- [7] Příspěvatelé Wikipedie. *Object-relational mapping* [online]. 2009. [cit. 6.12.2009]. Dostupné z: <[http://en.wikipedia.org/wiki/Object-relational\\_mapping](http://en.wikipedia.org/wiki/Object-relational_mapping)>.
- [8] Příspěvatelé Wikipedie. *Telnet* [online]. 2010. [cit. 12.5.2010]. Dostupné z: <<http://cs.wikipedia.org/wiki/Telnet>>.
- [9] Příspěvatelé Wikipedie. *Unified Modeling Language* [online]. 2010. [cit. 12.5.2010]. Dostupné z: <<http://cs.wikipedia.org/wiki/UML>>.
- [10] SLAVÍK, P. Grammars and Rewriting Systems as Models for Graphical User Interfaces. *Cognitive Systems*. 1997, 4, 3/4, s. 381–399.
- [11] The Internet Assigned Numbers Authority. *IANA IPv4 Address Space Registry* [online]. 2010. [cit. 19.5.2010]. Dostupné z: <<http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml>>.
- [12] web:cstug. CSTUG —  $\mathcal{C}\mathcal{S}\mathcal{T}\mathcal{E}\mathcal{X}$  Users Group — hlavní stránka. <http://www.cstug.cz/>, stav z 2.3.2009.
- [13] web:infodp. K336 Info — pokyny pro psaní diplomových prací. <https://info336.felk.cvut.cz/clanek.php?id=400>, stav ze 4.5.2009.

- [14] web:infogs. Knihovna Grafické skupiny.  
<http://www.cgg.cvut.cz/Bib/library/>, stav z 30.8.2001.
- [15] web:ipe. Grafický vektorový editor pro práce vhodný pro práci L<sup>A</sup>T<sub>E</sub>Xem.  
<http://tclab.kaist.ac.kr/ipe/>, stav z 4.5.2009.
- [16] web:latexdocweb. L<sup>A</sup>T<sub>E</sub>X — online manuál.  
<http://www.cstug.cz/latex/lm/frames.html>, stav ze 4.5.2009.
- [17] web:latexwiki. Wiki Books L<sup>A</sup>T<sub>E</sub>X.  
<http://en.wikibooks.org/wiki/LaTeX/>, stav z 3.4.2009.
- [18] ŽÁRA, J. – BENEŠ, B. – FELKEL, P. *Moderní počítačová grafika*. Computer Press s.r.o, Brno, 1st edition, 1998. In Czech.  
ceske norme), ale je to nejprehlednejsi.

## Příloha A

# Testování zaplnění stránky a odsazení odstavců

**Tato příloha nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.**

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?

Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili? Určitě existuje nějaká pěkná latinská věta, která se k tomuhle testování používá, ale co mají dělat ti, kteří se nikdy latinsky neučili?









## Příloha B

# Pokyny a návody k formátování textu práce

**Tato příloha samozřejmě nebude součástí vaší práce. Slouží pouze jako příklad formátování textu.**

Používat se dají všechny příkazy systému L<sup>A</sup>T<sub>E</sub>X. Existuje velké množství volně přístupné dokumentace, tutoriálů, příruček a dalších materiálů v elektronické podobě. Výchozím bodem, kromě Googlu, může být stránka CSTUG (Czech Tech Users Group) [12]. Tam najdete odkazy na další materiály. Většinou dostačující a přehledně organizovanou elektronikou dokumentaci najdete například na [16] nebo [17].

Existují i různé nadstavby nad systémy T<sub>E</sub>X a L<sup>A</sup>T<sub>E</sub>X, které výrazně usnadní psaní textu zejména začátečníkům. Velmi rozšířený v Linuxovém prostředí je systém Kile.

### B.1 Vkládání obrázků

Obrázky se umísťují do plovoucího prostředí **figure**. Každý obrázek by měl obsahovat **název** (`\caption`) a **návěští** (`\label`). Použití příkazu pro vložení obrázku `\includegraphics` je podmíněno aktivací (načtením) balíku `graphicx` příkazem `\usepackage{graphicx}`.

Budete-li zdrojový text zpracovávat pomocí programu **pdflatex**, očekávají se obrázky s příponou **\*.pdf**<sup>1</sup>, použijete-li k formátování **latex**, očekávají se obrázky s příponou **\*.eps**.<sup>2</sup>

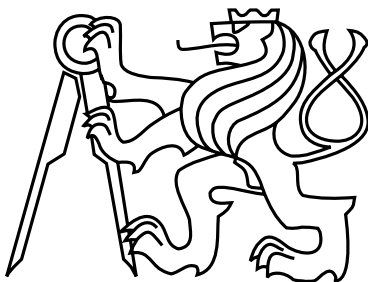
Příklad vložení obrázku:

```
\begin{figure}[h]
\begin{center}
\includegraphics[width=5cm]{figures/LogoCVUT}
\caption{Popiska obrazku}
\label{fig:logo}
```

---

<sup>1</sup>pdflatex umí také formáty PNG a JPG.

<sup>2</sup>Vzájemnou konverzi mezi snad všemi typy obrázku včetně změn velikostí a dalších vymožeností vám může zajistit balík ImageMagic (<http://www.imagemagick.org/script/index.php>). Je dostupný pod Linuxem, Mac OS i MS Windows. Důležité jsou zejména příkazy `convert` a `identify`.



Obrázek B.1: Popiska obrázku

DTD	construction	elimination
	in1 A B a:sum A B in1 A B b:sum A B	case([_:A] a) ([_:B] a) ab:A case([_:A] b) ([_:B] b) ba:B
+	do_reg:A -> reg A	undo_reg:reg A -> A
*, ?	the same like   and + with empty_el:empty	the same like   and + with empty_el:empty
R(a,b)	make_R:A->B->R	a: R -> A b: R -> B

Tabulka B.1: Ukázka tabulky

```
\end{center}
\end{figure}
```

## B.2 Kreslení obrázků

Zřejmě každý z vás má nějaký oblíbený nástroj pro tvorbu obrázků. Jde jen o to, abyste dokázali obrázek uložit v požadovaném formátu nebo jej do něj konvertovat (viz předchozí kapitola). Je zřejmě vhodné kreslit obrázky vektorově. Celkem oblíbený, na ovládání celkem jednoduchý a přitom dostatečně mocný je například program Inkscape.

Zde stojí za to upozornit na kreslicí programe Ipe [15], který dokáže do obrázku vkládat komentáře přímo v latexovském formátu (vzroce, stejné fonty atd.). Podobné věci umí na Linuxové platformě nástroj Xfig.

Za pozornost ještě stojí schopnost editoru Ipe importovat obrázek (jpg nebo bitmap) a krelit do něj latexovské popisky a komentáře. Výsledek pak umí exportovat přímo do pdf.

## B.3 Tabulky

Existuje více způsobů, jak sázet tabulky. Například je možno použít prostředí `table`, které je velmi podobné prostředí `figure`.

Zdrojový text tabulky B.1 vypadá takto:

```

\begin{table}
\begin{center}
\begin{tabular}{|c|l|l|}
\hline
\textbf{DTD} & \textbf{construction} & \textbf{elimination} \\
\hline
 $\mid$  & \verb+in1|A|B a:sum A B+ & \verb+case([_:A]a)([_:B]a)ab:A+\\
& \verb+in1|A|B b:sum A B+ & \verb+case([_:A]b)([_:B]b)ba:B+\\
\hline
 $\$$  & \verb+do_reg:A -> reg A+ & \verb+undo_reg:reg A -> A+\\
\hline
 $*,?$  & the same like  $\mid$  and  $\$$  & the same like  $\mid$  and  $\$$ \\
& with \verb+empty_el:empty+ & with \verb+empty_el:empty+\\
\hline
R(a,b) & \verb+make_R:A->B->R+ & \verb+a: R -> A+\\
& & \verb+b: R -> B+\\
\hline
\end{tabular}
\end{center}
\caption{Ukázka tabulky}
\label{tab:tab1}
\end{table}
\begin{table}

```

## B.4 Odkazy v textu

### B.4.1 Odkazy na literaturu

Jsou realizovány příkazem `\cite{odkaz}`.

Seznam literatury je dobré zapsat do samostatného souboru a ten pak zpracovat programem bibtex (viz soubor `reference.bib`). Zdrojový soubor pro bibtex vypadá například takto:

```

@Article{Chen01,
  author   = "Yong-Sheng Chen and Yi-Ping Hung and Chiou-Shann Fuh",
  title    = "Fast Block Matching Algorithm Based on
              the Winner-Update Strategy",
  journal  = "IEEE Transactions On Image Processing",
  pages    = "1212--1222",
  volume   = 10,
  number   = 8,
  year     = 2001,
}

@Misc{latexdocweb,

```

```

author = "",
title = "{\LaTeX} --- online manuál",
note = "\verb|http://www.cstug.cz/latex/lm/frames.html|",
year = "",
}
...

```

**Pozor:** Sazba názvů odkazů je dána BibTeX stylem (`\bibliographystyle{abbrv}`). BibTeX tedy obvykle vysází velké pouze počáteční písmeno z názvu zdroje, ostatní písmena zůstanou malá bez ohledu na to, jak je napíšete. Přesněji řečeno, styl může zvolit pro každý typ publikace jiné konverze. Pro časopisecké články třeba výše uvedené, jiné pro monografie (u nich často bývá naopak velikost písmen zachována).

Pokud chcete BibTeXu napovědět, která písmena nechat bez konverzí (viz `title = "{\LaTeX} --- online manuál"` v předchozím příkladu), je nutné příslušné písmeno (zde celé makro) uzavřít do složených závorek. Pro přehlednost je proto vhodné celé parametry uzavírat do uvozovek (`author = "..."`), nikoliv do složených závorek.

Odkazy na literaturu ve zdrojovém textu se pak zapisují:

```

Podívejte se na \cite{Chen01},
další detaily najdete na \cite{latexdocweb}

```

Vazbu mezi soubory `*.tex` a `*.bib` zajistíte příkazem `\bibliography{}` v souboru `*.tex`. V našem případě tedy zdrojový dokument `thesis.tex` obsahuje příkaz `\bibliography{reference}`.

Zpracování zdrojového textu s odkazy se provede postupným voláním programů `pdflatex <soubor>` (případně `latex <soubor>`), `bibtex <soubor>` a opět `pdflatex <soubor>`.<sup>3</sup>

Níže uvedený příklad je převzat z dříve existujících pokynů studentům, kteří dělají svou diplomovou nebo bakalářskou práci v Grafické skupině.<sup>4</sup> Zde se praví:

```

...
j) Seznam literatury a dalších použitých pramenů, odkazy na WWW stránky, ...
Pozor na to, že na veškeré uvedené prameny se musíte v textu práce
odkazovat -- [1].
Pramen, na který neodkazujete, vypadá, že jste ho vlastně nepotřebovali
a je uveden jen do počtu. Příklad citace knihy [1], článku v časopise [2],
statí ve sborníku [3] a html odkazu [4]:
[1] J. Žára, B. Beneš;, and P. Felkel.
    Moderní počítačová grafika. Computer Press s.r.o, Brno, 1 edition, 1998.
    (in Czech).

```

<sup>3</sup>První volání `pdflatex` vytvoří soubor s koncovkou `*.aux`, který je vstupem pro program `bibtex`, pak je potřeba znovu zavolat program `pdflatex (latex)`, který tentokrát zpracuje soubory s příponami `.aux` a `.tex`. Informaci o případných nevyřešených odkazech (cross-reference) vidíte přímo při zpracovávání zdrojového souboru příkazem `pdflatex`. Program `pdflatex (latex)` lze volat vícekrát, pokud stále vidíte nevyřešené závislosti.

<sup>4</sup>Několikrát jsem byl upozorněn, že web s těmito pokyny byl zrušen, proto jej zde přímo necituji. Nicméně příklad sám o sobě dokumentuje obecně přijímaný konsensus ohledně citací v bakalářských a diplomových pracích na KP.

- [2] P. Slavík. Grammars and Rewriting Systems as Models for Graphical User Interfaces. *Cognitive Systems*, 4(4--3):381--399, 1997.
- [3] M. Haindl, Š. Kment, and P. Slavík. Virtual Information Systems. In *WSCG'2000 -- Short communication papers*, pages 22--27, Pilsen, 2000. University of West Bohemia.
- [4] Knihovna grafické skupiny katedry počítačů:  
<http://www.cgg.cvut.cz/Bib/library/>

... abychom výše citované odkazy skutečně našli v (automaticky generovaném) seznamu literatury tohoto textu, musíme je nyní alespoň jednou citovat: Kniha [18], článek v časopisu [10], příspěvek na konferenci [1], [www odkaz](#) [14].

Ještě přidáme další ukázkou citací online zdrojů podle české normy. Odkaz na wiki o frameworkch [3] a ORM [7]. Použití viz soubor `reference.bib`. V seznamu literatury by nyní měly být živé odkazy na zdroje. V `reference.bib` je zcela nový typ publikace. Detaily dohledal a dodal Petr Dlouhý v dubnu 2010. Podrobnosti najdete ve zdrojovém souboru tohoto textu v komentáři u příkazu `\thebibliography`.

## B.4.2 Odkazy na obrázky, tabulky a kapitoly

- Označení místa v textu, na které chcete později čtenáře práce odkázat, se provede příkazem `\label{navesti}`. Lze použít v prostředích `figure` a `table`, ale též za názvem kapitoly nebo podkapitoly.
- Na návěští se odkážeme příkazem `\ref{navesti}` nebo `\pageref{navesti}`.

## B.5 Rovnice, centrováná, číselovaná matematika

Jednoduchý matematický výraz zapsaný přímo do textu se vysází pomocí prostředí `math`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$`.

Kód `$ S = \pi * r^2 $` bude vysázen takto:  $S = \pi * r^2$ .

Pokud chcete nečíselované rovnice, ale umístěné centrováně na samostatné řádky, pak lze použít prostředí `displaymath`, resp. zkrácený zápis pomocí uzavření textu rovnice mezi znaky `$$`. Zdrojový kód: `$$$ S = \pi * r^2 $$$` bude pak vysázen takto:

$$S = \pi * r^2$$

Chcete-li mít rovnice číselované, je třeba použít prostředí `equation`. Kód:

```
\begin{equation}
  S = \pi * r^2
\end{equation}
```

```
\begin{equation}
  V = \pi * r^3
\end{equation}
```

je potom vysázen takto:

$$S = \pi * r^2 \quad (\text{B.1})$$

$$V = \pi * r^3 \quad (\text{B.2})$$

## B.6 Kódy programu

Chceme-li vysázet například část zdrojového kódu programu (bez formátování), hodí se prostředí *verbatim*:

```

(* nickname2 *)
Lego> Refine in1
      (do_reg (nickname1 h));
Refine by in1 (do_reg (nickname1 h))
  ?4 : pcddata
  ?5 : pcddata
      (* surname2 *)
Lego> Refine surname1 h;
Refine by surname1 h
  ?5 : pcddata
      (* email2 *)
Lego> Refine undo_reg (email1 h);
Refine by undo_reg (email1 h)
*** QED ***
```

## B.7 Další poznámky

### B.7.1 České uvozovky

V souboru `k336_thesis_macros.tex` je příkaz `\uv{}` pro sázení českých uvozovek. „Text uzavřený do českých uvozovek.“



## Příloha C

# 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 D

# 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 E

# Instalační a uživatelská příručka

Tato příloha velmi žádoucí zejména u softwarových implementačních prací.



## Příloha F

# Obsah přiloženého CD

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

Může vypadat například takto. Váš seznam samozřejmě bude odpovídat typu vaší práce. (viz [13]):



Obrázek F.1: Seznam přiloženého CD — příklad

Na GNU/Linuxu si strukturu přiloženého 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.