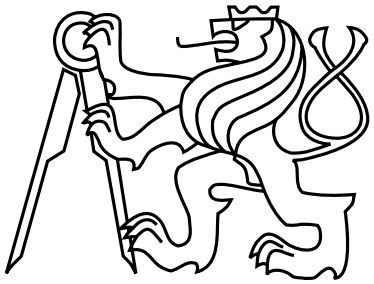
Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním 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 Linux

Tomáš Pitřinec

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

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Softwarové inženýrství

26. 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 25. 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

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Jednou z laboratorních úloh předmětu Počítačové sítě (Y36PSI) na FELu je postavení sítě mezi několika linuxovými a ciscovými počítači. Studenti, kteří tento úkol plní, nemají často s takovou činností žádnou osobní zkušenost, a tak během úlohy řeší různé banální problémy, kterým by se mohli vyhnout, kdyby měli možnost zkusit si nastavit podobou síť již před samotnou laboratorní úlohou. Mohl by se jim hodit simulátor, který by jednoduše spustili na svém počítači a na kterém by si mohli nastavování síťových parametrů na linuxu a ciscu zkusit. Právě návrhem a implementací takového síťového simulátoru počítačů s OS Linux se zabývá tato bakalářská práce.

1.1 Cíle práce

Cílem práce je v programovacím jazyce Java SE navrhnout a implementovat aplikaci, která umožní vytvoření virtuální počítačové sítě, pro potřeby předmětu Y36PSI. Z pohledu uživatele by aplikace měla vypadat stejně jako reálná síť. Uživatel spustí aplikaci v konsoli a pak se pomocí telnetu připojí k jejím jednotlivým virtuálním počítačům, podobně jako protokolem ssh k počítačům s OS Linux. Aplikace bude podporovat příkazy potřebné ke konfiguraci síťových rozhraní (`ifconfig`, `ip address`), směrování (`route`, `ip route`) a překladu adres (`iptables -t nat`). Pro ověření správnosti konfigurace sítě budou implementovány příkazy `ping` a `traceroute`.

Nastavenou konfiguraci sítě bude možné uložit do souboru a zase ji ze souboru načíst. Uživatel bude mít možnost vytvářet libovolné sítě s libovolným počtem počítačů typu linux nebo cisco tak, že infrastrukturu sítě napíše do konfiguračního souboru a pak ji z něho načte.

1.2 Struktura práce

Ve druhé kapitole této práce popisuji některé již existující simulátory a zamýšlím se nad jejich využitím pro výukové předměty. Ve třetí kapitole provádím analýzu aplikace. V následujících dvou kapitolách popisuji její realizaci, nejprve realizaci aplikace jako celku a poté realizací jednotlivých příkazů, které jsou v simulátoru implementovány. V závěru práce navrhuji některá vylepšení vytvořeného simulátoru a rozebírám jeho testování.

Kapitola 2

Existující řešení

Existujících síťových simulátorů je na trhu mnoho, avšak ne všechny splňují podmínky, aby byly jednoduše využitelné pro předmět PSI. Některé nejsou vůbec tvořené pro výukové účely. Nenašel jsem mnoho simulátorů, které by zároveň podporovaly síťové prvky s OS Linux i Cisco IOS a byly vhodné k výukovým účelům.

2.1 Packet tracer

Tento velmi známý program nemohu v této kapitole vynechat. Je to program přímo od společnosti Cisco, který věrně simuluje různé cisco switche a routery. V simulátoru je jen málo odchylek od skutečných zařízení[10]. Má grafické uživatelské rozhraní (obrázek 2.1, umí i zobrazovat pohyb paketů v síti. Pro potřeby předmětu Y36PSI má však velké nevýhody. Tou první je, že je volně dostupný pouze členům Cisco Networking Academy. Druhá nevýhoda je, že tento simulátor neumožňuje simulovat i počítače s OS Linux.

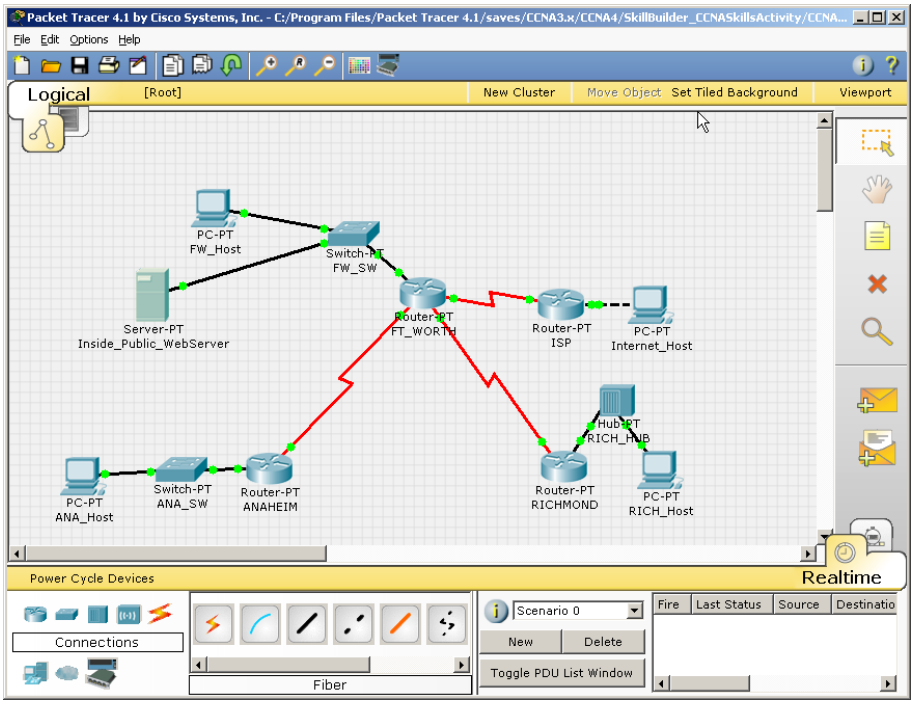
2.2 AdventNet Simulation Toolkit

„AdventNet Simulation Toolkit je kompletní grafický softwarový simulátor zařízení a sítí s velkou podporou mnohých protokolů, Cisco IOS zařízení a jiných zařízení jako např. různé Linux, či Windows 2000 servery.“[13] Argumentem proti použití tohoto programu je především jeho cena, plná časově neomezená verze stojí od $995 do $14995 [1]. K disposici je zkušební třicetidení verze.

2.3 Simulační software Omnet++

„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.“[13]. Systém se však hodí spíše pro simulaci zatížení sítě a pro simulaci síťových protokolů. K výukovým účelům není příliš vhodný kvůli své složitosti.

[1] k 23.5.2010

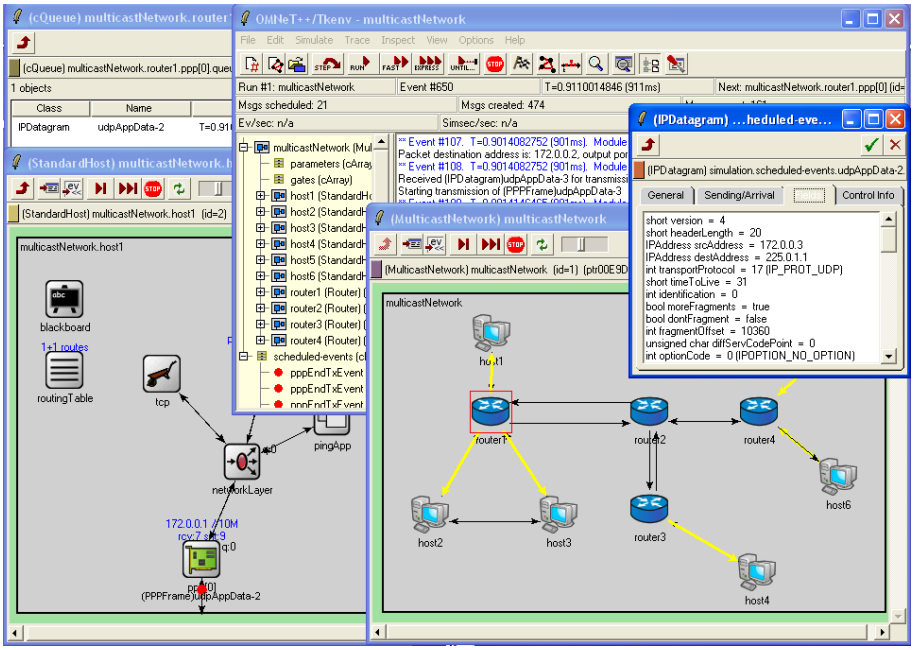


Obrázek 2.1: Packet tracer

Více se tímto simulátorem zabýval Bc. Jan Michek v rámci své diplomové práce Emulátor počítačové sítě [4].

2.4 Závěr

Tvůrci simulačních programů dávají většinou přednost simulaci sítí založených na síťových prvcích ood firmy Cisco. Podporují-li i počítače s linuxem, ani o tom moc nepíší. Většina simulátorů je studentům nedostupná kvůli své ceně. Studenti předmětu PSI budou simulátor potřebovat pravděpodobně jen několikrát za semestr a je nesmyslné, aby si škola, nebo sami studenti kupovali kvůli několika použitím licenci.



Obrázek 2.2: Omnet++

Kapitola 3

Analysa aplikace

V této kapitole se zabývám analysou a návrhem aplikace jako celku. Shrnuji a analysuji požadavky, diskutuji zvolený jazyk a uživatelské rozhraní, navrhuji architekturu aplikace a odhaduji její náročnost. Analysa jednotlivých částí simulátoru je popisována společně s těmito částmi.

3.1 Požadavky na aplikaci

Nejprve shrnu všechny požadavky na mojí aplikaci.

3.1.1 Funkční požadavky

1. Vytvoření počítačové sítě založené na počítačích OS Linux.
2. Aplikace umožňuje konfiguraci rozhraní pomocí příkazů ifconfig a ip addr.
3. Aplikace obsahuje funkční směrování a umožňuje jeho nastavování pomocí příkazů route a ip route.
4. Aplikace implementuje překlad adres
5. Aplikace podporuje ukládání a načítání do/ze souboru.
6. Pro ověření správnosti jsou implementovány příkazy ping a traceroute.
7. K jednotlivým počítačům aplikace je možné se připojit pomocí telnetu.
8. Pomocí telnetu bude možno se připojit zároveň k více virtuálním počítačům.
9. Pomocí telnetu bude možno připojit se k jednomu počítači vícekrát najednou.

3.1.2 Nefunkční požadavky

1. Aplikace bude multiplatformní - alespoň pro operační systémy Windows a Linux
2. Aplikace musí být spustitelná na běžném[1] studentském počítači.
3. Aplikace by měla být co nejvěrnější kopií reálného počítače s Linuxem.

3.2 Analysa požadavků

3.2.1 Připojení pomocí telnetu

Jedním z funkčních požadavků mé aplikace je možnost připojit se k jednotlivým virtuálním počítačům pomocí protokolu telnet. Tento požadavek vypadá jednoduše, pokud pod pojmem Telnet chápeme jednoduchý protokol na přenos textových dat. Takový protokol ovšem neumožňuje doplňování příkazů a jejich historii, což je pro práci s počítačem, byť virtuálním, obrovské omezení. Oproti tomu, implementovat telnet protokol, jako NVT[2], kde se posílá a potvrzuje každý napsaný znak, by překračovalo rozsah této bakalářské práce. Můj kolega nalezl program rlwrap, který poskytuje historii příkazů a jejich doplňování na straně klienta. Funguje v linuxu, na windowsu jen pod cygwinem. Toto druhé řešení bude o dost jednodušší a rozhodli jsme se ho realisovat, i když pro uživatele bude nevýhodou spouštění přes cygwin. I tak ovšem základní požadavek, že s aplikací bude možno komunikovat pomocí telnetu, zůstane zachován, uživatel ovšem přijde o komfort, který mu nabízí možnost doplňování, editace a historie příkazů.

3.2.2 Podobnost simulátoru se skutečným linuxem

Aby byl simulátor využitelný pro výukové účely, musí být dostatečně podobný skutečnému linuxu, aby uživatel mohl věřit, že to, co funguje v simulátoru, bude fungovat i na skutečném linuxu a naopak. K tomu bude stačit implementovat jen ty příkazy, kterými se nastavují síťové parametry, a jen v takovém rozsahu, jaký je pro tyto výukové účely potřeba. Budu tedy implementovat příkazy `ifconfig\verb`, `route`, `ping` a `traceroute`, z příkazu `ip` stačí implementovat jeho podpříkazy `addr` a `route`. Pro potřeby nastavení překladu adres je potřeba implementovat malou část příkazu `iptables`. Aby uživatel mohl nastavovat některé hodnoty souborů v adresáři `/proc`, implementuji ve velmi omezené míře i příkazy `cat` a `echo`, ovšem jen pro tyto soubory. Pro ukončení spojení bude implementován příkaz `exit`. Pro potřeby simulátoru ale není potřeba implementovat kompletní příkazy `ifconfig` nebo `ip`, ale jen tu jejich část, kterou se nastavují parametry rozhraní, jako IP, maska a další. O ostatních parametrech pak vetšinou simulátor vypíše, že ve skutečnosti sice existují, ale simulátorem zatím nejsou podporované.

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

[2] NVT – Network Virtual Terminal, česky: Síťový virtuální terminál; poskytuje standardní rozhraní příkazové řádky

3.2.3 Počet simulovaných počítačů

Na laboratořích Y36PSI studenti konfigurují 4 počítače, náš simulátor by měl zvládnout simulovat síť o 10 počítačích. Více ani není potřeba, pro výukové účely studenti pravděpodobně nebudout konfigurovat více počítačů.

3.3 Programovací jazyk a uživatelské rozhraní

3.3.1 Programovací jazyk

Aplikaci jsme se rozhodli programovat v programovacím jazyku Java z několika důvodů. Java je programovací jazyk, který nabízí velký programátorský komfort, stabilitu a zároveň možnost vytvořené aplikace používat pod různými operačními systémy, což je další z nefunkčních požadavků. Tento jazyk navíc disponuje hotovými knihovnami pro práci se sítí v balíčku java.net. Dalším důvodem je také to, že s programováním aplikací v Javě mám zatím asi největší zkušenosti.

3.3.2 Uživatelské rozhraní

Jak plyne ze zadání, uživatel se přihlašuje k jednotlivým virtuálním počítačům pomocí programu telnet, nemusím tedy vytvářet žádného speciálního klienta. S aplikací samotnou nebude uživatel nijak pracovat, jenom ji spustí se správným konfiguračním souborem a případně číslem výchozího portu, dále již bude nastavovat pouze jednotlivé virtuální počítače pomocí telnetu. Pro takovou aplikaci je nejlepším uživatelským rozhraním příkazová řádka, vytváření grafického uživatelského rozhraní by nemělo smysl.

3.4 Návrh architektury

Aplikace se bude skládat ze dvou vrstev. Komunikační vrstva by měla zajišťovat síťovou komunikaci s klientem, tedy odesílání a přijímání textových dat. Z velké části bude převzata z jiné práce, kterou jsme kdysi dělali jako domácí úkol na předmět Y36PSI. Aplikační vrstva bude tvořena samotnou virtuální sítí. Tyto vrstvy však od sebe nebudou striktně odděleny. Nejprve si rozebereme druhou vrstvu.

3.4.1 Virtuální síť

Virtuální počítačová síť, kterou bude aplikace simulovat, má poskytovat především tyto funkcionality:

- Možnost konfigurace jednotlivých síťových prvků.
- Posílání paketů mezi síťovými prvky.

Skutečná počítačová síť se skládá ze síťových prvků různých druhů. Stejně tak i virtuální síť se bude skládat ze síťových prvků, které budou interně reprezentovány objekty.

3.4.1.1 Síťové prvky

V laboratořích předmětu Y36PSI studenti nastavují pouze PC nebo směrovače na 3. (síťové) vrstvě ISO/OSI modelu[3]. Síťové prvky pracující na 2. vrstvě ISO/OSI modelu[4], switche a bridge se v laboratořích vůbec neuvažují. Proto i ve své práci uvažuji jediný druh síťových prvků - počítače s OS Linux.

3.4.1.2 Posílání paketů

Virtuální síť musí umět posílat virtuální pakety, aby uživatel pomocí příkazů `ping` nebo `traceroute` zjistil, jestli virtuální síť správně nakonfiguroval. Posílání paktů bude vnitřně realisováno vzájemným voláním metod virtuálních počítačů, které si mezi sebou budou předávat objekty typu paket. Tyto metody zřejmě bude vhodné rozdělit tak, aby odpovídaly jednotlivým vrstvám ISO/OSI modelu.

3.4.2 Komunikační vrstva

Komunikační vrstva simulátoru bude zajišťovat spojení aplikace s klientem. Z tohoto pohledu bude simulátor klasickým síťovým serverem, který poslouchá na několika portech, přijímá spojení a zpracovává je. Uživatel bude po síti konfigurovat jednotlivé virtuální počítače, proto každý virtuální počítač musí poslouchat na jednom portu. Pro obsluhu této komunikace bude vytvořeno několik tříd. Aby mohl simulátor poslouchat na více portech najednou, bude nutné vytvořit více vláken, každý virtuální počítač tedy poběží v samostatném vláknu. Jak plyne z posledního funkčního požadavku, musí jeden virtuální počítač umět zpracovat i více spojení najednou, jako i na reálný linuxový počítač je možné se připojit k několika jeho terminálům pomocí protokolu ssh nebo telnet. Proto bude nutné, aby vlákno, které poslouchá na portu, pro příchozí spojení vytvořilo jiné vlákno, které spojení obslouží, a samo dále poslouchalo na určeném portu.

3.5 Odhad náročnosti aplikace

Aplikace nebude mít žádné uživatelské rozhraní, nebude přistupovat do žádné database a její datové struktury budou pravděpodobně poměrně jednoduché. Proto pro virtuální síť o deseti počítačích by spotřeba paměti by neměla překročit požadavky pro běžné aplikace v Javě. Simulátor by mohl potřebovat odhadem 10-30MB bez načteného prostředí JRE.

3.6 Odhad složitosti práce a jejího průběhu

Celý projekt by měl být v rozsahu zhruba 10000 řádků kódu a měl by být dokončen do konce dubna 2010.

[3] 3. vrstva ISO modelu, tzv. síťová vrstva, zajišťuje spojení mezi jakýmikoliv 2 uzly sítě.

[4] 2. vrstva ISO/OSI modelu, tzv. spojová nebo linková vrstva, zajišťuje spojení mezi dvěma sousedními systémy.

Kapitola 4

Implementace virtuální sítě

V této kapitole se zabývám analysou a implementací jednotlivých částí aplikace. Nejdříve popisuji architekturu aplikace jako celku, dále rozebírám analýzu a implementaci třídy `IpAdresa`, implementaci virtuálního počítače, analýzu a implementaci routovací tabulky a analýsu a implementaci posílání paketů. Nezabývám se zde analysou a implementací jednotlivých příkazů, vzhledem k rozsáhlosti tohoto tématu jsem ho vyčlenil do zvláštní kapitoly, která následuje za touto kapitolou.

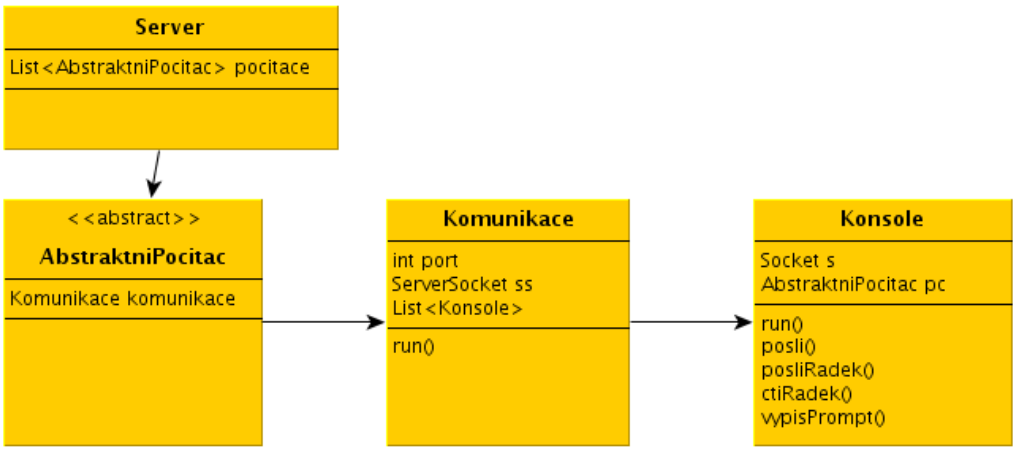
4.1 Popis architektury aplikace

Aplikace se skládá ze dvou vrstev. První, komunikační vrstva zajišťuje veškerou síťovou komunikaci s klientem, druhá, aplikační vrstva reprezentuje virtuální síť, která je simulována.

4.1.1 Komunikační vrstva

Komunikační vrstva byla z velké části přejata z úkolu na předmět Y36PSI, který jsme programovali na podzim roku 2008. Síťová komunikace s klientem je velmi jednoduchá, server s klientem si navzájem posílají jen textová data, tzn. klient posílá serveru příkazy v textové podobě a server na ně odpovídá.

Hlavní třídou komunikační vrstvy je třída `Komunikace`. Ta se stará o veškerou komunikaci virtuálního počítače s uživatelem. Je potomkem třídy `Thread`. Běží ve vlastním vlákně, které se startuje v jejím konstruktoru, a poslouchá na portu, který jí byl zadán. Pro každé nové příchozí spojení vytvoří instanci třídy `Konsole`, která spojení obslouží, aby komunikce mohla dále poslouchat na portu a zpracovávat další spojení. Třída `Konsole` je také potomkem třídy Thread. Obsluhuje jedno telnetové připojení. Drží si instanci třídy `ParserPrikazu` z balíčku `Prikazy` (o něm v následující kapitole). Přijímá textová data od uživatele až po enter (sekvence `\r\n`), tedy vlastně načítá data po řádcích. Každý řádek, který uživatel pošle, předá parseru na zpracování a pak sama pošle uživateli prompt. Parseru poskytuje metody pro posílání textových dat uživateli. Pro uživatele tak komunikace s touto konsolí vypada stejně jako práce s příkazovou řádkou na skutečném počítači.



Obrázek 4.1: Komunikační vrstva

4.1.2 Aplikační vrstva - virtuální síť

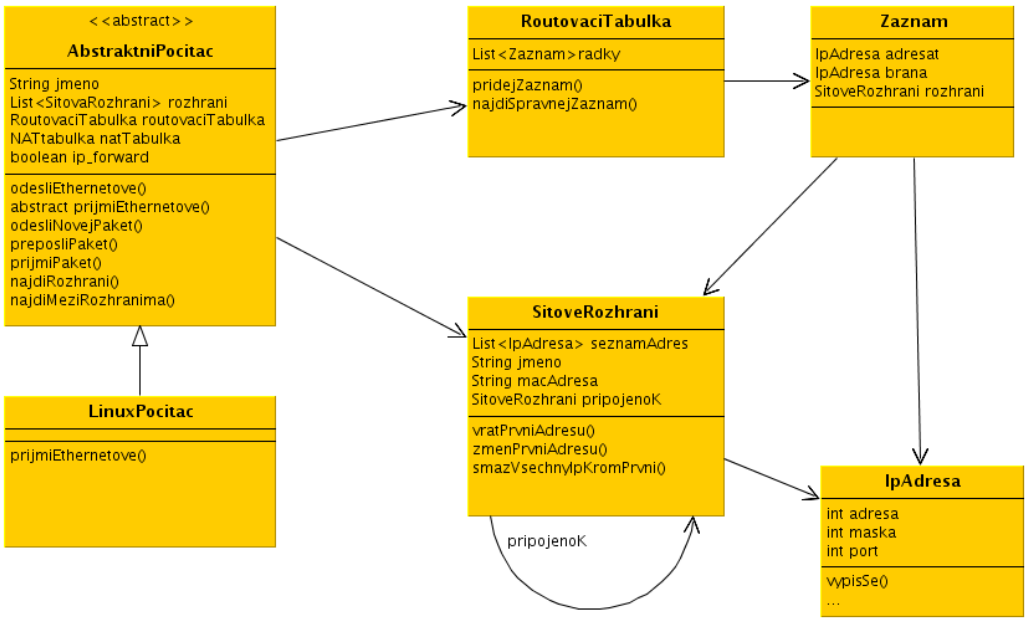
Před tím, než začnu popisovat implementaci jednotlivých komponent aplikační vrstvy, sluší se, popsat ji zhruba jako celek. Virtuální síť se skládá z virtuálních počítačů, což jsou objekty potomků abstraktní třídy `AbstraktniPocitac`. Ty si mezi sebou voláním svých metod předávají pakety, což jsou objekty třídy `Paket`. Virtuální počítače mají síťová rozhraní, což jsou objekty třídy `SitovaRozhrani`. Jejich pomocí jsou počítače mezi sebou propojeny (více o infrastruktuře v 4.4)

4.2 IP adresa

Třída `IpAdresa` je sice jen jednou z mnoha tříd, vzhledem k jejímu významu ji ale v následujících odstavcích popíšu podrobněji.

4.2.1 Analýza

Protože simulátor se zabývá především simulací síťové vrstvy ISO/OSI modelu, je síťová adresa počítače, tzv. IP adresa velmi často používanou datovou strukturou, pro kterou se vyplatí mít speciální třídu. Ta se v v aplikaci jmenuje `IpAdresa` a patří do balíčku `datoveStruktury`. Je používána jako parametr síťového rozhraní, jako prefix v routovací tabulce, jako zdrojová a cílová adresa v paketech. Při bližším pohledu je zřejmé, že kontext jejího použití se v těchto případech částečně liší. Například pro posílání paketů je nutné, aby paket obsahoval zdrojovou a cílovou adresu i s portem. Port by samozřejmě nemusel být součástí adresy, to se ale ukázalo jako jednoduší a pro posílání paketů přehlednější možnost. Pro IP adresu jako parametr rozhraní je naopak port zcela nesmyslný parametr, nutně ale potřebuje parametr pro síťovou masku, která je naproti tomu nesmyslná pro posílání paketů. Paket posílám na IP adresu, ne na adresu s maskou.



Obrázek 4.2: Architektura aplikační vrstvy - virtuální síť

4.2.2 Vnitřní reprezentace

Přes tyto rozdíly jsem se rozhodl vytvořit pro IP adresu jednu třídu, která má parametry `adresa`, `maska` a `port`, přičemž pro danou situaci nepotřebné parametry prostě ignoruji. Protože Java neobsahuje žádný 32-bitový bezeznaménkový datový typ, jsou parametry adresa a maska vnitřně reprezentovány 32-bitovým integerem, který ale obsahuje bity skutečné adresy, jeho číselná hodnota není důležitá. Operace s nimi se provádí především pomocí bitových operátorů. Parametr port je normální integer.

4.2.3 Veřejné metody

`IpAdresa` má konstruktory, aby ji bylo možné vytvořit ze `Stringu`, s maskou zadanou jako `String`, `Integer` nebo v jednou řetězci s adresou. Adresu je možné převést na String nebo porovnat s jinou adresou mnoha různými způsoby, například jen podle adresy, adresy s portem, adresy s maskou nebo čísla sítě. `IpAdresa` umí vrátit své číslo sítě nebo broadcast jako jinou `IpAdresu`. O těchto metodách se zde nerozepisuji podrobně, v kódu jsou dobře okomentované.

4.3 Virtuální počítač

Virtuální počítač je základním stavebním prvkem naší aplikace. Pracuje na obou jejích vrstvách. Na vrstvě komunikační přijímá a zpracovává příchozí spojení, na aplikační vrstvě, tj.

na vrstvě virtuální sítě přijímá, posílá a přeposílá pakety. Posíláním paketů se zabývám až v posledním odstavci této kapitoly, zde proberu komunikační vrstvu počítače a jeho rozhraní.

Protože linuxový a ciscový počítač, který dělal kolega, mají mnoho společného, vytvořil jsem abstraktní třídu `AbstraktniPocitac`, který je předkem počítačů obou typů. Třída `LinuxPocitac` má ale jen jednu metodu, která se týká posílání paketů, se jí zatím nezabývám.

Všechny virtuální počítače jsou vytvářeny v rámci inicialisace aplikace dle konfiguračního souboru na začátku jejícho běhu, za chodu aplikace není již možné další počítač přidat nebo nějaký odebrat. Pro komunikaci s uživatelem má každý počítač vlastní objekt třídy Komunikace. Počítač si drží seznam svých síťových rozhraní, svoji routovací tabulku a natovací tabulku. Má jediný konstruktor, kde je mu zadáno jméno (pro přehlednost) a port, na kterém má být dostupný pro uživatele.

4.3.1 Síťové rozhraní

Z hlediska infrastruktury sítě jsou základními prvky počítače jeho síťová rozhraní. Ty si počítač drží v seznamu. Jsou vytvořeny při parsovaní konfiguračního souboru a během běhu aplikace je nelze nijak měnit, přidávat nebo mazat. Třída `SitoveRozhrani` má svoje jméno a fysickou (mac) adresu. Protože v naší aplikaci není implementován ARP[1] protokol, mac adresa nemá jiný význam, než že je vypisována příkazy jako např. `ifconfig`.

Skutečné síťové rozhraní může mít více adres. Tato možnost však není v předmětu PSI využívána, proto jsem ji neimplementoval. Znamenalo by to totiž poměrně velké problémy v posílání paketů. Musel bych složitě zjišťovat, kdy se paket odešle s jakou síťovou adresou, pokud je jich na daném rozhraní více. Pro potřeby statického překladu adres (NAT) především na ciscovém routeru je ale nutné mít na rozhraní více adres[2]. Proto má třída `SitoveRozhrani` seznam IP adres, ale jeho první adresa je privilegovaná. Každý paket, který je přes dané rozhraní posílán, má jako odchozí adresu adresu právě první tohoto rozhraní. První adresa je vždy nastave, není-li nakonfigurována, je nastavena na `null`. Tuto jedinou adresu lze nastavovat a vypisovat. Ostatní adresy jsou přidávány jen pro potřeby statického natování. Pokud rozhraní nemá nastavenou žádnou adresu, je první (privilegovaná) adresa `null`.

4.4 Infrastruktura virtuální sítě

Poté, co jsem popsal implementaci virtuálního počítače, můžu popsat vnitřní reprezentaci infrastruktury virtuální sítě. Ta vychází z toho, že v simulátoru neuvažuji směrovače na linkové vrstvě ISO/OSI modelu (switche). To totiž znamená, že jedno síťové rozhraní počítače může být připojeno nejvýše k jednomu jinému síťovému rozhraní nějakého počítače. Infrastruktura takové sítě je tak jednoznačně uršena dvojicemi síťových rozhraní, které jsou mezi sebou propojeny kabelem. Třída `SitoveRozhrani` má proto parametr `pripojenoK`, který obsahuje odkaz na jiné síťové rozhraní, ke kterému je připojeno. Není-li rozhraní pripojeno, je tento

[1] 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 odesilatel zná pouze IP adresu. Pro odeslání prostřednictvím např. Ethernetu ale potřebuje znát cílovou ethernetovou adresu.[5]

[2] Více o překladu adres v bakalářské práci mého kolegy Stanislava Řeháka

parametr nastaven na `null`. Tato infrastruktura je samozřejmě načítána z konfiguračního souboru. Při vytváření infrastruktury aplikace ohlídá, aby rozhraní byla spojena obousměrně a správně. To jest, je-li rozhraní `A` připojeno k rozhraní `B`, musí být také rozhraní `B` připojeno k rozhraní `A`.

4.5 Routovací tabulka

Počítače směrují pakety podle tzv. routovací, neboli směrovací, tabulky. „Routovací tabulka je datový soubor uložený v RAM paměti, který je používán k uchovávání informací ohledně přímo připojených i vzdáleně připojených sítích. Její obsah napovídá routeru, kterým rozhraním je možno nejoptimálněji dosáhnout cílové sítě.“[3]. V této části se zabývám nejprve analýzou routovací tabulky na skutečném linuxu a potom popisuji její implementaci v simulátoru.

Třída `RoutovaciTabulka` měla být původně stejně použitelná pro linux i pro cisco. Až po tom, co jsem jí implementoval, kolega zjistil, že pro potřeby Cisca není tato třída bez úprav použitelná. Proto implementoval třídu `CiscoWrapper`, která obaluje třídu `RoutovaciTabulka` a dodává jí funkce potřebné pro cisco. To však není obsahem mojí práce.

4.5.1 Analýza routovací tabulky na skutečném počítači

4.5.1.1 Struktura tabulky

V řádcích routovací tabulky jsou záznamy pro jednotlivé sítě. Každý záznam má tyto parametry:

- adresát - IP adresa s maskou, pro kterou je tento záznam platný
- brána - IP adresa počítače, na který se má paket poslat. Tento sloupec nemusí být vždy vyplněn.
- příznaky - O těch více píšu v samostatné části.
- metrika - Jedno z kriterií priority.
- rozhraní - Rozhraní, přes které se paket posílá.

Parametr metrika není pro výukové účely potřeba, proto se jím již dále nezabývám.

Pro lepší představu zde vkládám routovací tabulku tak, jak je vypsána příkazem `route -n`:

```
Adresát         Brána           Maska           Přízn Metrik Odkaz  Užt Rozhraní
147.32.125.128  0.0.0.0         255.255.255.128 U     1      0        0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U     1000   0        0 eth0
0.0.0.0         147.32.125.129  0.0.0.0         UG    0      0        0 eth0
```

4.5.1.2 Adresát

V hořejším výpisu tabulky pomocí příkazu `route` se adresáta týkají 2 sloupce, sloupec Adresát a sloupec Maska, ve kterém jsou vypsány masky k IP adresám uvedeným ve sloupci Adresát. Tyto IP adresy s maskami jsou vždy číslem sítě a reprezentují všechny adresy, které do této sítě patří. Tak například adresát 0.0.0.0/0 reprezentuje úplně všechny IP adresy, adresát 147.32.125.128/25 reprezentuje adresy v rozmezí 147.32.125.128 až 147.32.125.255, adresát 1.1.1.1/32 reprezentuje jedinou adresu 1.1.1.1 a adresát 192.168.1.0/24 reprezentuje všechny adresy, které začínají byty 192.168.1.x. Adresáta 147.32.125.128/24 nelze zadat, protože číslo této sítě je 147.32.125.0/24.

4.5.1.3 Příznaky

Záznam routovací tabulky má několik příznaků. Má vždy minimálně jeden příznak, může mít ale všechny 3 příznaky najednou. Zde je jejich popis:

- Příznak `U` znamená, že záznam obsahuje rozhraní. Protože záznam bez vyplněného rozhraní není možné zadat, má tento příznak každý záznam.
- Záznam má příznak `G`, jestliže je vyplněn sloupec brána.
- Příznak H znamená, že adresátem daného záznamu je jeden počítač, tzn. adresát má masku 255.255.255.255.

Příznak `H` jen informuje, že adresát není sítí ale jediným počítačem, není tedy nijak důležitý. Podle příznaků existují 2 typy záznamů, záznamy s příznakem `U` a záznamy s příznakem `UG`. Tyto typy se liší jak při přidávání nových záznamů do routovací tabulky, tak při posílání paketu podle tohoto záznamu. Posílá-li počítač paket podle záznamu U, není z tohoto záznamu zřejmé, jakému sousednímu počítači (na linkové vrstvě) se má paket poslat. Počítač se tedy pokusí poslat paket přímo na cílovou IP adresu uvedenou v paketu. Posílá-li se paket podle záznamu UG, posílá se na adresu brány uvedenou v záznamu. Více se této problematice věnuji v kapitole o posílání paketů.

4.5.1.4 Přidávání záznamů a jejich řazení

Routovací tabulka nesmí obsahovat 2 stejné záznamy. Za stejné záznamy se považují záznamy, které mají stejného adresáta včetně masky, stejné rozhraní a stejnou bránu.

Záznam typu `U` lze přidat vždycky. Záznam typu `UG` lze přidat jen pod podmínkou, že jeho brána je v okomžiku přidání dosažitelná záznamem typu `U`. Tím je možné dosáhnout zajímaveho chování: Když do routovací tabulky přidám defaultní routu [3] záznamu typu U, můžu pak přidat routu na jakoukoliv síť v internetu se záznamem UG. Když potom smažu původní defaultní routu, můžu posílat pakety pouze na tu síť s příznakem `UG` a na počítač v mé síti paket neodešlu. Zde uvádím příklad:

[3] Defaultní routa je záznam platný pro ccelý internet, jeho adresátem je 0.0.0.0/0

```
root: /home/neiss# route add default eth0
root: /home/neiss# route add -net 89.190.94.0/24 gw 89.190.94.1
root: /home/neiss# route del default
root: /home/neiss# route
Směrovací tabulka v jádru pro IP
Adresát         Brána           Maska           Přízn Metrik Odkaz  Užt Rozhraní
89.190.94.0     89.190.94.1     255.255.255.0   UG    0      0        0 eth0
root: /home/neiss# ping -c1 89.190.94.58
PING 89.190.94.58 (89.190.94.58) 56(84) bytes of data.
64 bytes from 89.190.94.58: icmp_seq=1 ttl=53 time=14.1 ms

--- 89.190.94.58 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 14.141/14.141/14.141/0.000 ms
root: /home/neiss# ping -c1 147.32.125.129 # toto je adresa brány, přes kterou šel minul
connect: Network is unreachable
```

Tento pokus funguje ale jen tehdy, pokud moje brána, v tomto případě 147.32.125.129 je cisco (viz část o posílání paketů).

Záznamy se v tabulce řadí podle masky adresáta. Nahoře jsou záznamy s nejdelší maskou, tzn. záznamy nejkonkrétnější. Pokud vkládám více záznamů se stejnou maskou, chová se routovací tabulka naprosto nepředvídatelně, což je vidět na následujícím příkladě:

```
node-4:/home/dsn# route add -net 1.1.5.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.6.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.7.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.8.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.9.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.10.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.11.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.12.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.13.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.14.0/25 dev eth0
node-4:/home/dsn# route add -net 1.1.15.0/25 dev eth0
node-4:/home/dsn# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
1.1.10.0        *               255.255.255.128 U     0      0        0 eth0
1.1.11.0        *               255.255.255.128 U     0      0        0 eth0
1.1.8.0         *               255.255.255.128 U     0      0        0 eth0
1.1.9.0         *               255.255.255.128 U     0      0        0 eth0
1.1.14.0        *               255.255.255.128 U     0      0        0 eth0
1.1.6.0         *               255.255.255.128 U     0      0        0 eth0
1.1.15.0        *               255.255.255.128 U     0      0        0 eth0
1.1.7.0         *               255.255.255.128 U     0      0        0 eth0
1.1.12.0        *               255.255.255.128 U     0      0        0 eth0
```

```
1.1.13.0        *               255.255.255.128 U     0      0        0 eth0
1.1.5.0         *               255.255.255.128 U     0      0        0 eth0
```

Podle jakého algoritmu jsou nové záznamy zařazovány je mi opravdu záhadou.

4.5.1.5 Mazání záznamů

Smazat je možno jakýkoliv záznam tabulky. Pro mazání záznamů je potřeba zadat správně minimálně adresáta záznamu. Pokud pak existuje více záznamů se zadanými parametry, smaže se první z nich. Smazání jakéhokoliv záznamu nijak neovlivní ostatní záznamy.

4.5.1.6 Použití pro směrování

Ke směrování paketu se použije první záznam odpovídající cílové adrese. To znamená, že bude-li v routovací tabulce dané adrese odpovídat více záznamů, použije se ten nejvíce nahoře. Protože záznamy jsou řazeny podle délky síťové masky, je vrácený záznam ten nejkonkrétnější.

4.5.2 Implementace routovací tabulky v simulátoru

Routovací tabulka je imlementována třídou `RoutovaciTabulka`, jejíž odkaz si drží `AbstraktniPocitac` a podle ní směruje pakety.

4.5.2.1 Vnitřní reprezentace

Tabulka je vnitřně reprezentována seznamem objektů typu `Zaznam`, který reprezentuje jeden záznam, tj. řádek tabulky. Záznam routovací tabulky má v simulátoru jen tyto parametry: adresát, brána a rozhraní, které fungují tak, jak bylo popsáno v odstavci o analyse. Parametry adresát a brána jsou typu `IpAdresa`, parametr rozhraní je typu `SitoveRozhrani`. Parametr záznamy není vůbec potřeba. Příznak `U` musí mít záznam vždy, příznak `H` má právě tehdy, když adresát má masku 255.255.255.255, a příznak `U` má záznam právě tehdy, když má vyplněnou položku brána, proto ani ten není potřeba.

4.5.2.2 Přidávání, mazání a řazení záznamů

Záznamy jsou přidávány pomocí 2 metod se stejným názvem `pridejZaznam` ale jinými parametry. Jedna přidává záznam typu `U`, druhá, která má navíc paramatr brána, záznam typu `UG`. U obou se kontroluje, jestli tabulka již stejný záznam neobsahuje, u té druhé se navíc kontroluje dosažitelnost brány, jak bylo popsáno v analýze.

Záznamy se samozřejmě řadí podle masky jako v reálné tabulce, záznamy se stejnou maskou se ale vloží vždy nad původní záznam. Tak jsou novější záznamy vždy nahoře. Zmatečné řazení reálné tabulky jsem samozřejmě neimplementoval.

Pro mazání má `RoutovaciTabulka` metodu `SmazZaznam`, funguje stejně jako na reálném počítači.

Navíc obsahuje `RoutovaciTabulka` ještě metodu `pridejZaznamBezKontrol`, která je využívána při vytváření počítače z konfiguračního souboru, jinde se nepoužívá.

4.5.2.3 Použití při směrování

K samotnému směrování slouží metoda `najdiSpravnejZaznam`, která vrací celý řádek routovací tabulky. Funguje stejně jako na reálném počítači.

4.6 Posílání paketů

Posílání paketů v naší aplikaci slouží k tomu, aby uživatel pomocí příkazů `ping` a `traceroute` mohl ověřit, zda síť správně nakonfiguroval. Bez této části by naší aplikaci nebylo možné nazvat síťovým simulátorem.

4.6.1 Teoretický rozbor referenčního modelu ISO/OSI

„Referenční model ISO/OSI vypracovala organizace ISO jako hlavní část snahy o standardizaci počítačových sítí.“[11]. Dle tohoto modelu probíhá síťová komunikace v sedmi vrstvách, z níž každá poskytuje přesně definované funkce a komunikuje jen s vrstvou sousední. Každá vrstva má svůj formát přenášených dat, obyčejně dělených do bloků. Pro moji aplikaci jsou důležité vrstvy 2 - 4, tzn. spojová, síťová a transportní vrstva.

4.6.1.1 Spojová vrstva

Spojová nebo linková vrstva[4] „poskytuje spojení mezi dvěma sousedními systémy.“[11]. Tuto vrstvu zajišťuje na skutečné síti v laboratoři technologie Ethernet[6], ke zjištění fysické adresy sousedního systému se používá protokol ARP[5]. Sousednímy systémy se zde rozumí počítače zapojené do stejné sítě. Blok dat na linkové vrstvě se nazývá rámec. Vzhledem k tomu, že simulátor neobsahuje žádné switche, zajištuje tato vrstva v naší aplikaci spojení mezi 2 počítači propojenými kabelem.

4.6.1.2 Síťová vrstva

Síťová vrstva „se stará o směrování v síti a síťové adresování. Poskytuje spojení mezi systémy, které spolu přímo nesousedí.“[11] Zajišťuje spojení meze jakýmikoliv dvěma uzly sítě. Obyčejně je realizována protokolem IP[5]. Blok dat na síťové vrstvě se nazává paket.

4.6.1.3 Transportní vrstva

Transportní vrstva „zajišťuje přenos dat mezi koncovými uzly“[11]. Pro potřeby příkazů ping a traceroute je tato vrstva realisována protokolem ICMP[6]. Blok dat se v transportní vrstvě nazývá datagram.

[4] Více v [9]
[5] Internet Protocol
[6] Internet Control Message Protocol, více v [7]

4.6.1.4 Datové bloky

Datový blok jakékoliv vrstvy se skládá z hlavičky, která obsahuje režijní informace té vrstvy, a z datové části, která obsahuje samotná data, ale i hlavičky vyšších vrstev. Tak například datagram protokolu ICMP je obalen hlavičkou IP na síťové vrstvě a hlavičkou protokolu Ethernet na spojové vrstvě.

4.6.2 Implementace třídy Paket

Pro můj síťový simulátor by bylo nesmyslné implementovat posílání paketů včetně jejich zabalování do datových bloků ruzných vrstev, tak jak je to popsáno v předchozím odstavci 4.6.1.4. Proto jsem vytvořil třídu `Paket`, která obsahuje všechny potřebné informace z datových bloků všech tří vrstev. `Paket` má parametry síťové vrstvy, jako zdrojovou a cílovou adresu a `ttl`, a parametry transportní vrstvy, jako typ a kód protokolu ICMP. Pro snadnější implementaci příkazu ping má parametr `cas`, kam se ukládá náhodně generovaný čas běhu paketu, který vypisuje příkaz ping. Protože na jednom virtuálním počítači může běžet více příkazů `ping` najednou, nese paket i odkaz na příkaz, který ho poslal, aby ho tento příkaz mohl také po jeho návratu zpracovat.

Typy a kódy ICMP paketů jsou označeny stejně jako ve skutečnosti:

- typ 0 - ozvěna (icmp reply) - odpověď na požadavek icmp request
- typ 3 - vyslaný paket nemohl být doručen
- typ 8 - žádost o ozvěnu (icmp request)
  - kód 0 - network unreachable (nedosažitelná síť)
  - kód 1 - host unreachable (nedosažitelná adresa)
- typ 11 - `ttl` vypršelo

4.6.3 Chování reálného počítače při posílání paketů

4.6.3.1 Transportní vrstva - protokol ICMP

V souboru `/proc/sys/net/ipv4/icmp_echo_ignore_all` je nastaveno, jestli počítač odpovídá na dotazy icmp reply. Pokud je v tomto souboru `0`, počítač na dotazy odpovídá. Toto je defaultní nastavení, proto jsem v simulátoru tento problém vůbec neřešil a počítač odpovídá na icmp request vždy.

4.6.3.2 Síťová vrstva - protokol IP

Přeposílání paketů

Počítač preposílá pakety jenom tehdy, pokud je v souboru `/proc/sys/net/ipv4/ip_forward` jednička, jinak ne. Toto nastavení ale již není defaultní, proto ho musím v simulátoru implementovat. Proměnná `ip_forward` je parametrem virtuálního počítače a je nastavována pomocí příkazu `echo`.

Směrování paketů

Pakety jsou směrovány podle routovací tabulky, kde se vybere první záznam odpovídající cílové adrese paketu, jak je popsáno v 4.5.1.6. Pokud v routovací tabulce nebyl nalezen žádný záznam pro cílovou adresu paketu, pošle se na jeho zdrojovou adresu paket `icmp_net_unreachable`[7].

ttl

Každý prvek, který pracuje i na síťové vrstvě sníží procházejícím paketům hodnotu `ttl`. Pokud po tomto snížení dosáhne hodnota `ttl` nuly, pošle počítač na zdrojovou adresu paketu zprávu `icmp_ttl_exceeded`[8].

Next hop

Předtím než síťová vrstva předá odeslání paketu linkové vrstvě, musí pro tento paket zjistit tzv. next hop, neboli sousední adresu. „Next hop je sousední směrovač, na který je paket poslán nebo přeposlán z daného směrovače na své cestě k cíli.“[?] Tato adresa je velmi důležitá pro linkovou vrstvu, která paket posílá právě na tuto adresu. Pokud je paket směrován podle záznamu typu `UG` (viz 4.5.1.3), je adresa next hop uvedena ve sloupci brána routovací tabulky. Pokud je paket směrován podle záznamu typu `U`, je adresa next hop cílová adresa paketu. Podle routovací tabulky

Adresát	Brána	Maska	Přízn	Metrika	Rozhraní
147.32.125.128	0.0.0.0	255.255.255.128	U	0	eth0
0.0.0.0	147.32.125.129	0.0.0.0	UG	0	eth0

je pro pakety směrované podle prvního záznamu next hop rovná jejich cílové adrese. Pro pakety směrované podle druhého záznamu je next hop `147.32.125.128`. Záznamy typu `U` jsou tak používány pro počítače v mé síti, které jsou dosažitelné přímo bez jakéhokoliv mezilehlého směrovače. Záznamy `UG` jsou používány pro počítače, na které je paket posílán přes jeden nebo více směrovačů.

4.6.3.3 Linková vrstva

Na linkové vrstvě se rámce přeposílají jen mezi sousedními počítači. Běžně ji zajišťuje protokol ethernet. IP adresu sousedního počítače, na kterou má rámec poslat, next hop, dostane od síťové vrstvy, musí ji přeložit na fysickou (MAC[9]) adresu, což dělá protokolem ARP. Protokol ARP vyšle ethernetový rámec na všechny okolní počítače žádost, která obsahuje zadanou IP adresu. Pokud některý z počítačů má tokovou IP adresu, původnímu počítači pošle zpátky svoji fysickou adresu a ten pak může odeslat paket. Aby počítač nemusel fysickou adresu zjišťovat při odesílání každého rámce, ukládá si v datové struktuře nazvané ARP tabulka záznamy s IP adresami, které již dříve překládal.

Pokud počítač nemůže linkovou vrstvou odeslat rámec (paket), například proto, že počítač s takovou adresou na síti neexistuje, pošle původnímu odesílateli ICMP paket `net unreachable`[10].

[7] Tj ICMP paket typu 3 kódu 0
[8] Tj. icmp paket typu 11
[9] Media Access Control
[10] Tj. ICMP paket typu 3 kódu 1

Počítač s operačním systémem linux odpovídá na ARP dotazy jen tehdy, když má požadovanou IP adresu nastavenou na svém rozhraní. V tom se liší od Cisca, které na ARP dotaz odpoví i v případě, že adresu na svém rozhraní nemá, ale ví, kam má paket dále směřoav, tzn. má pro požadovanou adresu záznam v routovací tabulce, a naopak na ARP dotaz neodpoví v případě, že nemá v routovací tabulce záznam pro IP adresu, odkud dotaz přišel. Pro lepší pochopení přepisuji podmínku ještě v jazyce booleanovských výrazů v javě:
Cisco odpoví na ARP dotaz právě tehdy, když:
Má záznam v routovací tabulce pro počítač, který ARP dotaz posílá `&&` (Má nastavenou požadovanou IP adresu `nebo` Má záznam v routovací tabulce pro cílovou adresu posílaného paketu)

4.6.3.4 Zajímavá zjištění

Při analýze chování linuxového počítače v síťové komunikaci jsem zjistil několik zajímavých a aspoň pro mě překvapujících faktů. Například 2 počítače v síti spolu můžou komunikovat i tehdy, mají-li nastaveny IP adresy z úplně jiných sítí. Stačí totiž, mají-li v routovací tabulce jeden na druhého záznam. Počítače z obrázku 4.3 spolu opravdu komunikují, jak je vidět na následujícím výpisu.

Obrázek 4.3: Funkční síť se „špatnými“ adresami

```
node-1:/home/dsn# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:40:F4:B7:F0:32
          inet addr:192.168.1.1  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::240:f4ff:feb7:f032/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43 errors:0 dropped:0 overruns:0 frame:0

node-1:/home/dsn# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=1 ttl=64 time=0.942 ms
64 bytes from 10.0.0.1: icmp_seq=2 ttl=64 time=0.208 ms

--- 10.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1003ms
```

```
rtt min/avg/max/mdev = 0.191/0.447/0.942/0.350 ms
```

4.6.4 Implementace v simulátoru

Pakety jsou mezi počítači posílány vzájemným voláním metod, které si mezi sebou předávají objekt typu `Paket`. Všechny tyto metody jsou ve třídě `AbstraktniPocitac`, protože to jsou právě virtuální počítače, které si mezi sebou pakety posílají. Jen metoda `prijmiEthernetove` je sice v `AbstraktniPocitac` zadeklarována a implementována v jeho potomcích.

Metody virtuálního počítače pro posílání paketů jsem bylo rozumné implementovat dle vrstev. Je to přehlednější, než kdybych měl jednu metodu pro přijmutí paketu a je to dobré i vzhledem k tomu, že linux a cisco se v některých případech liší a to jen na některých vrstvách. Metody jedné vrstvy tak volají jen metody vrstvy sousední, jako na reálné síti jedna vrstva komunikuje jen s vrstvami sousedními.

4.6.4.1 Linková vrstva

ARP protokol na zjišťování fysických adres nemusel být implementován, protože v síti nejsou žádné switche a v linkové vrstvě tak není potřeba žádné směrování. Aby ale byly splněny podmínky doručení nebo nedoručení paketu uvedené v 4.6.3.3, byly vytvořeny metody `odesliEthernetove` a `prijmiEthernetove`, které si mezi sebou předávají pakety tak, aby byly tyto podmínky splněny. Metoda `odesliEthernetove` je volána nějakou metodou síťové vrstvy. Pokouší se odeslat paket tak, že zavolá metodu `prijmiEthernetove` nějakého jiného počítače. Metoda `prijmiEthernetove` na linuxu přijme paket jen tehdy, pokud souhlasí očekávaná adresa, tj. adresa next hop odesílacího počítače. Přijme tedy paket právě tehdy, když by skutečný počítač odpověděl na ARP dotaz. Pokud se metodě `odesliEthernetove` nepovede paket odeslat proto, že ho metoda `prijmiEthernetove` odmítla, pošle odesílateli pomocí metody `posliNovejPaketOdpoved` zprávu `icmp host unreachable`. Pokud metoda `prijmiEthernetove` paket přijme, zavolá metodu `prijmiPaket` síťové vrstvy.

4.6.4.2 Síťová vrstva

Na síťové vrstvě si pakety předávají metody `odesliNovejPaket`, `preposliPaket` a `prijmiPaket`. Tyto metody směrují pakety dle routovací tabulky a provádí překlad adres podle natovací tabulky. Všechny pakety přijímá metoda `prijmiPaket`, která se nejdříve pokusí přeložit cílové adresy paketů. Pak rozhodne, je-li přijatý paket na tom počítači v cíli, nebo jestli se má dále přeposlat a případně zavolá metodu `preposliPaket` nebo paket nějak zpracuje, například odpoví na `icmp request`. Metoda `preposliPaket` přeposílá pakety, když má nastaveno `ip_forward`, a přeposílaným paketům snižuje ttl. Při odeslání se pokouší přeložit zdrojovou adresu paketu. Metoda `odesliNovejPaket` slouží k odesílání všech nových paketů vytvořených na tom počítači.

4.6.4.3 Transportní vrstva

Do transportní vrstvy patří více metod, které slouží k posílání různých typů ICMP paketů. Například `posliIcmpRequest`, `posliNetUnreachable` a další. Všechny tyto metody volají

metodu `odesliNovejPaket` ze síťové vrstvy.

Kapitola 5

Implementace příkazů

V této kapitole popisuji analýzu a realizaci linuxových příkazů, které jsou v simulátoru implementovány. Příkazy jsou nejrozsáhlejší částí mojí práce na simulátoru a nutno dodat, že také tou nejméně zábavnou. U každého příkazu jsem nejprve musel složitě zjišťovat, jak funguje a jak se chová v případě zadání nestandartních parametrů. Parsování příkazů bylo prací pořád stejnou a nezábavnou.

V této kapitole nejprve popisuji zpracování příkazové řádky, pak obecné principy pro zpracování všech příkazů a nakonec popisuji analýzu a implementaci konkrétních příkazů.

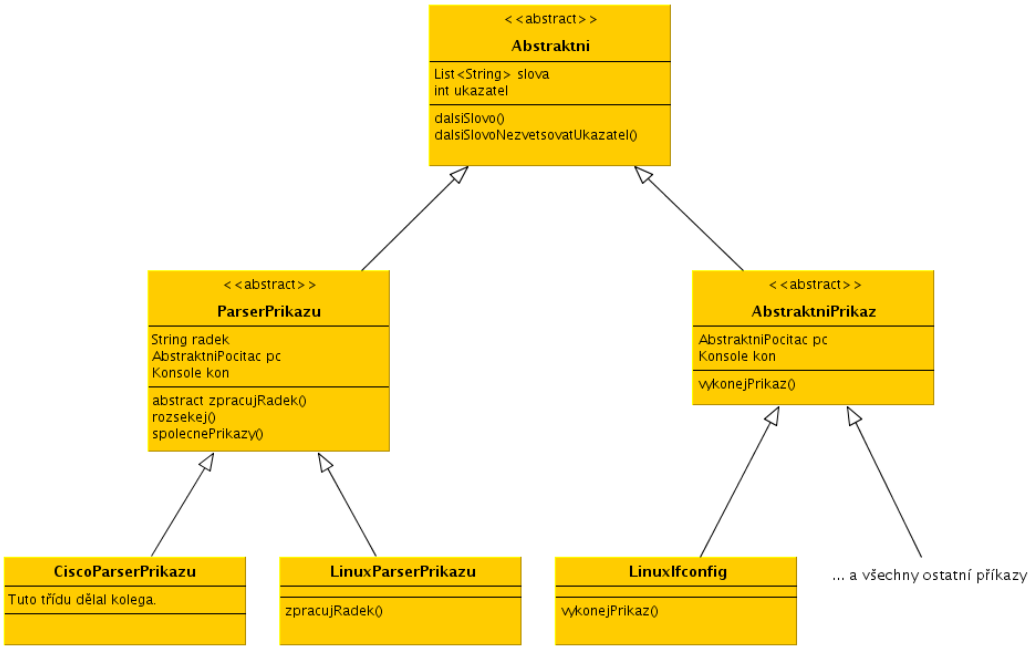
5.1 Popis zpracování příkazové řádky

Příkazovou řádku virtuálního počítače obsluhuje třída `Konsole` (více v 4.1.1). Ta načítá textová data posílaná uživatelem až do sekvence `\r\n`, kdy načítání zastaví a načtený řádek předá svému parseru. Každá `Konsole` má svůj vlastní parser příkazů, který zpracovává jen jí poslané řádky a na ně odpovídá. `Konsole` zavolá jeho metodu `zpracujRadek`, která řádek „rozseká“ na jedotlivá slova a pak podle prvního slova, které by mělo být názvem příkazu, spustí odpovídající příkaz. Příkazům `Konsole` poskytuje metody `posliRadek` a `posli`, pomocí kterých příkazy vypisují svůj výstup uživateli do konzole.

Architektura tříd pro zpracování příkazové řádky je celkem složitá, jak je vidět na obrázku 5.1. Všechny třídy mají jednoho předka, kterého jsme z nedostatku fantasie pojmenovali `Abstraktni`. Každý příkaz má svoji vlastní třídu, kterou `ParserPrikazu` vytvoří pro každé použití příkazu novou, jde tedy o použití návrhového vzoru singleton. V následujících odstavcích popisuji funkce jednotlivých tříd.

5.1.1 Třída Abstraktni

Tato třída je abstraktním předkem všech ostatních tříd sloužících ke zpracování příkazů. Obsahuje pro parsování velmi důležitou metodu `dalsiSlovo` a seznam slov jednoho řádku. Tato metoda postupně vrací jednotlivá slova ze seznamu. Je využívána skoro všemi potomky třídy `Abstraktni` k parsování řádky. Dále jsou v této třídě různé statické metody využívané jednotlivými příkazy, například metoda `zarovnej` na zarovnání slova v řetězci, metoda `cekej`, která čeká zadaný počet milisekund a další.



Obrázek 5.1: Třídy pro zpracování příkazové řádky

5.1.2 Třída ParserPrikazu

Instanci jednoho z potomků této třídy, tedy `LinuxParsePrikazu` nebo `CiscoParsePrikazu`, si vytváří při své inicialisaci třída `Konsole`, které slouží ke zpracování příkazové řádky. Jeho hlavní metodou je abstraktní `zpracujRadek`, která je implementována v jeho potomcích a zpracovává jeden řádek, který uživatel do konzole napsal. K tomu používá metodu `rozsekej`, jejímž autorem je můj kolega, která řádek zadaný jako `String` „rozseká“ na jednotlivá slova a to tak, že jako oddělovače bere bílé znaky, tj. mezery a tabulátory. Rozsekaná slova pak uloží do seznamu řetězců `slova`, který zdědila od svého předka třídy `Abstraktni`. Třída `LinuxParsePrikazu` podle prvního slova, které by mělo být názvempříkazu, vyvolá konstruktor třídy odpovídajícího příkazu. Nejdříve kontroluje, jestli příkaz není společným příkazem pro linuxový i ciscový počítač (příkaz `uloz` nebo `save`) a poté se snaží najít odpovídající linuxový příkaz. Jestliže uživatelem zadaný příkaz neexistuje nebo není implementován, vypíše `LinuxParsePrikazu` uživateli řetězec `bash: command not found` a zpracování řádku ukončí.

5.1.3 Třída AbstraktniPrikaz

Toto třída je potomkem třídy `Abstraktni` a předkem tříd všech příkazů. Nemá žádnou důležitou metodu, ale parametry `abstraktniPocitac pc` a `konsole kon`, což jsou odkazy na počítač, na kterém příkaz probíhá a na konsoli, která ho vyvolala.

5.2 Společné znaky příkazů

Příkazy, které jsem implementoval mají mnoho společného. Všechny jsou potomkem třídy `AbstraktniPrikaz`, který je potomkem třídy `Abstraktni`. Text, který byl uživatelem zadán, mají uložen v seznamu řetězců `slova`. V konstruktoru příkazu je postupně voláno několik privátních metod. Metodou `zparsujPrikaz` se nejprve parsuje vstup, většinou pomocí metody `dalsiSlovo` zděděné ze třídy `Abstraktni`. Při parsování se nastavují různé parametry příkazu, které uživatel zadal. Metoda `zkontrolujPrikaz` kontroluje, jestli jsou nastaveny správné parametry a jestli byli zadány všechny. Potom metoda `vykonejPrikaz` příkaz vykoná, to znamená, že změní konfiguraci počítače například u příkazu `ifconfig`, nebo že provede nějakou jinou akci, například u příkazu `ping` pošle několik paketů. Během těchto tří metod jsou ukládány případné chyby zadaného příkazu. Úplně nakonec příkaz vypíše případné chybové hlášení. Tento postup byl zvolen proto, že sladit správné pořadí vypisovaných chybových hlášení s postupem kontroly jednotlivých parametrů by bylo velmi náročné a nepřehledné. Příkazy svůj výstup posílají pomocí metod `posli` a `posliRade` třídy `Konsole`.

5.3 Příkaz ifconfig

5.3.1 Teoretický úvod

`ifconfig` (zkratka **i**nter**f**ace **config**urator) je utilita unixových systémů, která slouží ke konfiguraci, kontrole a vypsání informací o parametrech síťových rozhraní z příkazové řádky.[8]. Pomocí této utility lze například vypsat parametry síťových rozhraní, nastavit IP nebo mac adresy, nastavit masky nebo „nahodit“ nebo „schodit“ rozhrani. Kromě toho, že `ifconfig` nastavuje rozhraní, přidává nebo maže i záznamy v routovací tabulce, je-li to potřeba.

5.3.2 Rozsah implementace v simulátoru

V simulátoru musí `ifconfig` umět nastavovat a mazat adresy IP a nastavovat masky. Samozřejmě také musí vypsovat nastavené parametry rozhraní. Změna mac adresy naproti tomu v našem simulátoru není vůbec potřeba. Parser příkazu `ifconfig` je ale implementován i pro parametry `add`, `del`, `up` a `down`.

Zde uvádím příklady užití ifconfigu, které simulátor podporuje:

```
ifconfig eth0 vypíše parametry rozhraní eth0
ifconfig eth0 192.168.1.1 nastaví adresu na rozhraní eth0
ifconfig eth0 10.0.0.1/8 nastaví adresu s maskou na rozhraní eth0
ifconfig eth0 10.0.0.2 netmask 255.255.0.0 nastaví adresu s maskou na rozhraní eth0
```

5.3.3 Analýza ifconfigu na skutečném počítači

Oproti jiným příkazům má `ifconfig` pro simulaci jednu obrovskou nevýhodu: chybný parametr nezpůsobí ukončení příkazu, ale příkaz se dále provádí. Některé parametry je možné zadat vícekrát. Zjišťovat, jak se `ifconfig` chová, proto bylo mnohem náročnější, než u ostatních příkazů. Zde popisuji výsledky této práce.

IP adresu je možné příkazu zadat vícekrát, všechno, co nelze zparsovat jinak se považuje za IP adresu. Je-li zadáno více adres, použije se adresa, které předchází první špatně adrese. Například:

```
ifconfig eth0 1.1.1.1 2.2.2.2 3.3.3.3 nastaví se poslední adresa, tj. 3.3.3.3

ifconfig eth0 1.1.1.1 blablabla 3.3.3.3 nastaví se adresa před první špatnou adresou, tj. 1.1.1
blablabla: unknown host
ifconfig: ‘--help’ vypíše návod k použití.
```

Adresu je možné zadat i s délkou prefixu za lomítkem. Je-li zadáno více adres a u jedné z nich je zadána délka prefixu, nastaví se taková maska, i když se tato adresa nepoužije. Například

```
ifconfig eth0 1.1.1.1/1 2.2.2.2/2 3.3.3.3
```

nastaví adresu 3.3.3.3 s maskou 192.0.0.0, použije se tedy délka prefixu zadaná u druhé adresy. Adresa na rozhraní se maže nastavením adresy na 0.0.0.0.

Maska se nastavuje buď zadáním délky prefixu, jak je popsáno výše, nebo parametrem netmask. Zadá-li uživatel masku oběma způsoby, má přednost délka prefixu před parametrem netmask. Například

```
ifconfig eth0 2.2.2.2/2 netmask 255.255.0.0
```

nastaví masku 192.0.0.0. Při změně IP adresy se automaticky mění i maska a to tak, že se maska dopočítá podle třídy IP adresy. Je-li tedy nejprve zadána maska parametrem netmask a teprve potom IP adresa, parametr netmask nemá žádnou účinnost. Například

```
ifconfig eth0 netmask 255.255.255.0 1.1.1.1
```

nastaví adresu 1.1.1.1 s maskou 255.0.0.0.0, protože adresa 1.1.1.1 je třídy `A`.

Příkaz `ifconfig` promítá změny i do routovací tabulky. Změní-li se adresa na rozhraní, všecny záznamy na toto rozhraní se z tabulky vymažou. Při nastavení IP adresy se do routovací tabulky přidá záznam typu `U` pro síť, ze které je tato adresa. Například při nastavení adresy

```
ifconfig eth0 192.168.1.1/24
```

se do routovací tabulky přidá tento záznam:

```
Adresát         Brána           Maska           Přízn Metrik Odkaz  Užt Rozhraní
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

Při zadání chybného vstupu se vykoná jen správná část příkazu a o těch špatných se vypíší chybová hlášení. Ty jsou vypisována v tomto pořadí:

1. špatný přepínač - příkaz se neprovádí
2. špatná adresa - žádná další chybová hlášení se nevypisují

3. špatné rozhraní
4. zakázaná adresa
5. špatné adresy parametrů `add` nebo `del`
6. chyba v gramatice

Ostatními parametry se v tomto odstavci nezabývám, protože v simulátoru nejsou podporovány.

5.3.4 Implementace v simulátoru

V konstruktoru třídy `LinuxIfconfig` jsou popoředě volány tyto metody: `parsujPrikaz`, `zkontrolujPrikaz`, `vypisChybovyHlaseni` a `vykonejPrikaz`. Parsování se v tomto příkaze neprovádí s využitím metody `dalsiSlovo`, což by bylo dobré ještě upravit. Na rozdíl od skutečného ifconfigu se parsuje celý řádek, skutečný ifconfig parsování zastaví, když dojde ke špatnému přepínači nebo ke špatné adrese. Při parsování se parametry ukládají jako řetězce a ty jsou kontrolovány metodou `zkontrolujPrikaz`. V tomto příkazu jsou chybová hlášení vypisována před již před samotným provedením příkazu a to v pořadí uvedeném v předchozím odstavci. Příkaz `ifconfig` se metodou `vykonejPrikaz` nakonec provede tak, aby to co nejvíce odpovídalo pravidlům uvedným v 5.3.3. Parametry `add` a `del` jsou v parsetu a kontrole implementovány, ale metoda `vykonejPrikaz` je zatím ignoruje.

5.3.5 Možnosti dalšího vylepšení

Tento příkaz jsem implementoval jako první, kdy jsem s implementací příkazů ještě neměl zkušenosti, proto by se dal ještě v některých věcech vylepšit. Bylo by dobré přepsat metodu `parsujPrikaz` tak, aby využívala metodu `dalsiSlovo`, čímž by se zjednodušila, a dále ji předělat tak, aby se po špatné adrese již dále nic neparsovalo. Bylo by též dobré implementovat i parametry `up` a `down` s jejichž podporou se v analýze nepočítalo.

5.3.6 Známé odchylky

Zde uvádím známé odchylky implementace v simulátoru od skutečného příkazu na reálném počítači.
`ifconfig eth0 netmask 255.0.0.0 2.2.2.2/2` ve skutečnosti spadne kvůli prefixu za adresou, v simulátoru to ale projde.
`ifconfig eth0 blabla netmask 255.255.255.0` ve skutečnosti žádnou masku nenastaví, protože přestane parsovat už u blabla.
`ifconfig eth0 netmask 255.255.255.0 10.0.0.1` by měl nastavit masku na 255.0.0.0, v simulátoru se nastaví 255.255.255.0

5.4 Příkaz route

Příkaz route je utilita, která umožňuje administrátorovi vypisovat a měnit záznamy v routovací tabulce. Uvedu zde pár praktických příkladů:

```
route vypíše routovací tabulku
Směrovací tabulka v jádru pro IP
Adresát         Brána           Maska           Přízn Metrik Odkaz  Užt Rozhraní
192.168.1.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
0.0.0.0         192.168.1.2     0.0.0.0         UG    0      0        0 eth0

route add -net 10.0.0.0/8 gw 192.168.1.3 Přidá záznam typu UG pro síť 10.0.0.0/8 na bránu 192.1
route del -net 10.0.0.0/8 Odebere minule přidaný záznam
```

5.4.1 Analýza příkazu route na skutečném počítači

Oproti příkazu `ifconfig` má příkaz route jasně definovanou gramatiku a při jakékoliv chybě v příkazu se příkaz neprovádí, proto byla implementace tohoto příkazu podstatně jednoduší. Příkaz má 2 hlavní akce, `add` na přidávání záznamů a `del` na jejich mazání. Nápověda uvádí ještě akci `flush`, která ale není běžně podporována. Obě známé akce mají stejnou gramatiku. Parametr `-net` přidá nebo odebere záznam pro síť, parametr `-host` pro jednu adresu, tzn. záznam s maskou 255.255.255.255. Parametr `gw` slouží k nastavení brány přidávanému nebo odebíranému záznamu, parametr `dev` k nastavení rozhraní a parametr `netmask` k nastavení masky přidávanému nebo odebíranému záznamu. Maska může být stejně jako v příkazu `ifconfig` zadána parametrem `netmask` nebo pomocí délky prefixu za lomítkem, např. `10.0.0.0/16`. Při použití parametru `-net` musí být vždy maska nějakým způsobem zadána. Adresát, tzn. parametr `-net` nebo `-host` musí být vždy prvním parametrem akce, na pořadí dalších parametrů již nezáleží. Parametry `-host` a `dev` není nutno psát. Žádný parametr nemůže být zadán vícekrát. Akce `add` vyžaduje aspoň jeden z parametrů `gw` nebo `dev`. Narazí-li se při provádění přikazu na chybu, je vypsáno chybové hlášení a příkaz se ukončí, aniž by se vykonala jeho správná část.

5.4.2 Implementace v simulátoru

Implementoval jsem parametry `-host`, `-net`, `netmask`, `dev` a `gw`. Parser tohoto příkazu je psán za pomocí metody `dalsiSlovo` celkem přehledně, každý parametr se parsuje vlastní metodou. Parser též kontroluje správnost příkazu, vypisuje chybová hlášení a nastavuje parametry. Metoda `vykonejPrikaz` upravuje nebo vypisuje routovací tabulku příslušného počítače.

5.4.3 Odchylky

Akce `flush` sice není na normální počítači implementována, avšak v simulátoru jsem ji implementoval s tím, že simulátor vypíše varovné hlášení:

```
linux1:~# route flush
psi simulator: Flush normalne neni podporovano, ale routa se v simulatoru smaze.
Spravny prikaz je: "ip route flush all"
```

5.5 Příkaz ping

Tento příkaz posílá ICMP pakety typu `request` (žádost o odpověď) a přijímá pakety typu `ICMP reply`. Uživateli umožňuje prověřit funkčnost spojení mezi dvěma počítači v síti.

5.5.1 Analýza skutečného pingu

Příkazu musí uživatel zada cílovou adresu, na kterou se chce „dopingnout“. Dále je možné použít několik přepínačů. Přepínačem `-c` si uživael určí, kolik paketů chce poslat, není-li tento parametr zadán, posílaji se pakety pořád, dokud uživatel příkaz nezastací `Ctrl+C`. Parametrem `-s` je možne specifikovat velikost posílaného paketu, parametrem `-i` se specifikuje interval mezi odesláním dvou paketů. Parametrem `-t` je možné zadat `ttl` odchozích paketů. Parametr `-b` umožňuje posílat `icmp request` na broadcastovou adresu a parametr `-q` potlačuje výstup příkazu. Parametr `-b` není v simulátoru podporován. Například příkaz:

```
ping -c10 -i 0.1 -s1000 -t 23  192.168.2.2
```

pošle 10 paketů v intervalu 0,1 sekundy o velikosti 1000 bytů s ttl 23 na adresu 192.168.2.2. Na pořadí parametrů nezáleží, jejich hodnoty můžou být uvedeny bez mezery i s ní. U tohoto příkazu lze zadat přepínače i po IP adrese.

5.5.2 Implementace v simulátoru

Virtuální pakety, které si předávají virtuální počítače našeho simulátoru, si nesou i odkaz na příkaz, který je vyvolal. To proto, aby když počítači dorazí `icmp reply` počítač věděl, který příkaz je původcem tohoto paketu. Proto všechny příkazy, které v našem simulátoru posílají pakety jsou potomkem abstraktní třídy `AbstraktniPing` jejíž abstraktní metodu `zpracujPaket` volá přímo metoda počítače, který paket přijala. Tato třída má také metodu `aktualizujStatistiky`, která spočítá konečné statistiky příkazu.

Parser tohoto příkazu je podobný jako v ostatních příkazech, pomocí metody `dalsiSlovo` se parsují jednotlivé parametry. Příkaz nemá zvláštní metodu pro kontrolu pro výpis chybových hlášení. Metoda `vykonejPrikaz` posílá pomocí metody `posliIcmpRequest` třídy `AbstraktniPocitac` jednotlivé pakety a pak je metodou `zpracujPaket` zpracovává.

Čas odezvy vypisovaný tímto příkazem není skutečný, ale každý počítač procházejícímu paketu do parametru čas přičte náhodně vygenerovanou hodnotu a to tak, aby hodnoty byly přibližně stejné, jako jsou v laboratořích předmětu PSI.

5.5.3 Odchylky v implementaci

Protože program telnet, přes který se budou uživatelé simulátoru přihlašovat k počítačům neumí přeposílat `Ctrl+C`, musím v našem simulátoru mít nastavený výchozí počet odesílaných paketů na 4, aby uživatel tento příkaz mohl nějak zastavit. Toto je velký rozdíl oproti skutečnému pingu na linuxu, ale neměl jsem jinou možnost.

Parametr `-i`, kterým se nastavuje interval mezi odesláním dvou paketů, se musí v simulátoru zadat tak, aby mezi `-i` a hodnoutou byla mezera. Není tedy možné napsat `-i0.5`.

V simulátoru musí být oproti skutečnosti přepínače napsané před adresou. Ve skutečnosti to není nutné. Tento nedostatek ještě opravím.

5.6 Příkaz traceroute

Program traceroute slouží k analýze počítačové sítě. Vypisuje uzly (resp. směrovače) na cestě datagramů od zdroje až k zadanému cíli. Uzly jsou zjišťovány pomocí snížení hodnoty TTL v hlavičce datagramů.[12].

5.6.1 Popis činnosti

Příkaz posílá pakety ICMP request na cílovou adresu, kterým postupně zvyšuje `ttl`, dokud nedojte od cílového počítače paket ICMP reply. Počítače, které paket nemohou dále poslat, protože mu `ttl` vypršelo, posílají zdrojovému počítači zprávu ICMP time exceeded. Podle těchto zpráv příkaz zjistí, kterými uzly procházejí pakety k cílové adrese.

Příklad výpisu příkazu:

```
neiss: ~/smazat$ traceroute -n 89.190.94.58
traceroute to 89.190.94.58 (89.190.94.58), 30 hops max, 60 byte packets
 1  192.168.1.20  0.833 ms  1.257 ms  1.691 ms
 2  10.4.71.1  66.723 ms  67.245 ms  69.029 ms
 3  89.190.94.2  69.648 ms  70.088 ms  70.422 ms
 4  89.190.94.58  70.893 ms  71.261 ms  74.190 ms
```

5.6.2 Implementace

Třída příkazu `traceroute` dědí ze třídy `AbstraktniPing`, aby odkaz na ní mohl být posílán spolu s paketem. Příkaz nemá prakticky žádný parser, jediná povolená syntaxe tohoto příkazu je `traceroute <adresa>`. Pokud uživatel zadá neodpovídající vstup, vypíše se mu hlášení, jaká je jediná povolená syntaxe. Příkaz funguje na stejném principu jako příkaz skutečný, jak je popsáno v předchozím odstavci.

5.7 Příkaz exit

Příkaz `exit` je jedinou korektní možností ukončení relace. Zavolá metodu `ukonciSpojeni` třídy `Konsole`, čímž ukončí běh její metody `run` a tak se spojení ukončí.

5.8 Příkaz ip

Příkaz `ip` je plnohodnotnou náhradou za příkazy `ifconfig`, `route`, `arp`. Umožňuje však i pokrocilejší správu síťových nastavení[2]. Já jsem z tohoto příkazu ve svém simulátoru implementoval jeho dva základní podpříkazy `addr` pro nastavení parametrů rozhraní a `route` pro manipulaci se směrovací tabulkou.

5.8.1 Implementace

Příkaz `ip` je implementován třídou `LinuxIp`. Jeho podpříkazy `addr` a `route` ale byly implementovány samostatně, to znamená, že jsem pro každý podpříkaz vytvořil vlastní třídu. Třída `LinuxIp` parsuje přepínače a jméno podpříkazu. Je-li napsaný podpříkaz podporován, spustí odpovídající třídu, která pak funguje stejně jako samostatný příkaz. Není-li podporován, vypíše o tom hlášení.

Parsování příkazu `ip` má jednu zvláštnost. Není totiž nutné, aby uživatel psal všechny terminály, ale místo některých lze napsat třeba jen první písmeno, které terminál jednoznačně určuje. Tak například místo řádku

```
ip addr add 1.1.1.1 dev eth0
```

lze jednoduše napsat

```
ip a a 1.1.1.1 dev eth0
```

Tuto možnost podporuji i v simulátoru.

5.8.2 Podpříkaz addr

5.8.2.1 Analýza podpříkazu

Tento příkaz slouží především k přidávání a mazání adres na rozhraní. Má 4 akce: `add` na přidání adres, `del` na odebrání adresy, `flush` na smazání všech adres na rozraní a `show` na vypsání nastavených adres. Akce show je výchozí, není ji tedy nutno psát. Existující adresu na rozhraní není možno tímto příkazem změnit, je nutné starou IP odstranit a novou přidat.

5.8.2.2 Implementace

Tento příkaz je prováděn klasicky metodami `parsujPrikaz`, `zkontrolujPrikaz`, `vykonejPrikaz` a `vypisChybovyHlaseni`. Parser má pro každý parametr vlastní parsovací metodu. Kontrolovací metoda kontroluje zadané parametry podle akce, protože u každé akce mohou být jiné parametry.

5.8.2.3 Známé odchylky

V simulátoru není možné nastavit více adres a jednom rozhraní, když se o to uživatel snaží, vypíše mu simulátor varovné hlášení:

```
linux2:~# ip a a 1.1.1.1/22 dev eth0
psi simulator: Simulator nepodporuje vice adres na jednom rozhrani. Na rozhrani eth0 je
```

Akcím `flush` a `show` je možné zadat selektrory na vybrání mazaných nebo zobrazovaných adres. Ty nejsou v příkazu podporovány.

Na počítačích v laboratoři PSI projde příkaz

```
ip a a 600.0.0.1
```

tuto možnost však v simulátoru nepodporuji.

5.8.3 Podpříkaz route

5.8.3.1 Analýza

Příkaz `ip route` je alternativou k příkazu `route`, funguje prakticky stejně, ale má jinou, o něco jednoduší gramatiku. Na rozdíl od příkazu `route` neumožňuje přidat do routovací tabulky záznam na adresáta, který již v tabulce je. Má 5 akcí: `add` a `del` na přidávání a mazání záznamů, `show` a `flush` na vypsání a mazání všech záznamů a akci get, které se zadá cílová IP adresa a příkaz vypíše záznam ze směrovací tabulky, podle kterého by se směrovalo k této adrese. Všechny akce mají prakticky stejnou gramatiku, jen u akce `get` není povoleno zadat parametr `via`, což je obdoba parametru `gw` z příkazu `route`.

5.8.3.2 Implementace v simulátoru a její odchylky od skutečnosti

Způsob vykonávání tohoto příkazu je stejný jako u podpříkazu `addr`. Část metody `vykonejPrikaz` jsem mohl přejmout z příkazu `route`. V simulátoru nejsou podporovány selektory u akcí `show` a `flush`, u akce `get` není podporován parametr `dev`, o čemž se případně vypíše varovné hlášení.

5.9 Příkaz iptables

Příkaz `iptables` je utilita, která umožňuje nastavovat paketový filtr jádro operačního systému Linux. V simulátoru implementuji jen malou část rozsáhlého příkazu, která slouží k nastavování překladu adres.

Simulátor na linuxovém počítači podporuje jen tzv. statický NAT, který překládá všechny pakety odcházející přes rozhraní na jednu veřejnou IP adresu. V této práci se překladem adres nezabývám, tuto problematiku řešil především můj kolega, více informací je v jeho bakalářské práci.

5.9.1 Implementace

Příkaz je implementován k tomu, aby pomocí něj bylo možné nastavit statický překlad adres, vypsat jeho nastavení a vymazat ho, což se provádí například těmito příkazy:

```
iptables -t nat -I POSTORUTING -o eth2 -j MASQUERADE nastavení
iptables -t nat -L vypsání
iptables -t nat -D POSTROUTING 1 smazání
```

Parser je však implementován mnohem robustněji a umožňuje v budoucnosti přidat další možnosti nastavení. Oproti reálnému počítači není v našem simulátoru dovoleno nastavovat více pravidel pro překlad adres.

Kapitola 6

Uživatelské testování

Uživatelské testování prováděl tester za přítomnosti mé a mého kolegy z týmu. Jeho hlavním účelem bylo najít chyby v programu, na které jsem sám nepřišel. V této kapitole popisuji pouze testování těch částí systému, které jsem sám programoval. Testování částí, které dělal kolega, zde nepopisuji.

6.1 Průběh testování

6.1.1 Spuštění aplikace

Prvním úkolem testera bylo aplikaci spustit a zjistit, jak se s ní zachází. Toto činilo uživateli menší problémy, protože chybová hlášení vyhazovaná simulátorem nebyla dost jasná na to, aby uživatel pochopil, jak se má chyb vyvarovat. Spouštění aplikace je práce mého kolegy a ten chyby opravil.

6.1.2 Práce s aplikací

Tester měl za úkol zkonfigurovat síť z obrázku 6.1. Během konfigurace přišel na chyby, které zde popisuji, a zároveň také popisuji, jak byly opraveny.

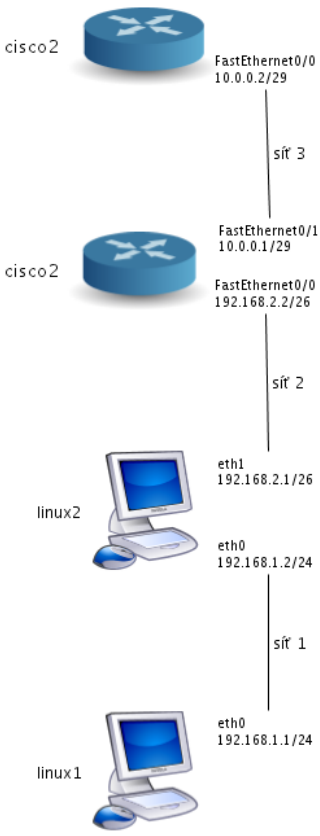
6.1.2.1 Výpisy

Hlášení o běhu simulátoru, která jsou vypisována na standartní výstup, se testerovi zdála velmi nepřehledná. Výpisy o tom, co server posílá jednotlivým klientům a co klienti posílají serveru, jsou pro uživatele nadbytečné a proto byly zrušeny. Ponechány byly jen výpisy o přihlášení klienta a o průchodu paketu, které byly testerovi užitečné pro diagnostiku sítě.

6.1.2.2 Příkaz ifconfig

Tester zkoušel změnit mac adresu rozhraní, což simulátor nepodporuje. Přidal jsem příkaz `help`, který popisuje, jaké varianty příkazů jsou podporovány.

Tester zkoušel příkaz `ifconfig --help`, který nefunguje. Chybu jsem opravil.



Obrázek 6.1: Testovací síť, kterou tester konfiguroval

6.1.2.3 Manuálové stránky

Tester zkusil zadat `man route`, přičemž simulátor vypsal, že příkaz neexistuje. Dodělal jsem příkaz `man`, který vypíše, že manuálové stránky nejsou implementovány a doporučí uživateli příkaz `help`.

6.1.2.4 Příkaz ping

Uživatel zadal příkaz `ping 192.168.1.1 -c5`. Zjistil, že parametr `-c` je ignorován, protože parser parsuje jen parametry před adresou. Parser jsem z důvodu velké časové tísně zatím nestihl opravit. Chyby je zmíněna přímo u popisu tohoto příkazu a bude opravena.

6.1.2.5 Soubor ipforward

Z důvodů snažšího testování jsem měl na soubor `/proc/sys/net/ipv4/ip_forward` nastaven alias `ip_forward`, který jsem zapomněl odstranit. Alias již byl odstraněn.

6.1.2.6 Příkaz route

Tester zkoušel příkaz `route --help`, který nefunguje. Do parseru jsem přidal parsování tohoto přepínače.

6.2 Závěr

V aplikaci bylo nalezeno několik, spíše drobnějších chyb především v parsování příkazů. Některé chyby již byly opraveny, některé teprve opravím a jistě existují chyby, zvláště v parserech, které nebyly ještě nalezeny. Ty budu muset opravit, až na ně uživatelé, budou-li kdy nějací, přijdou.

V simulátoru nebyly nalezeny žádné vážnější chyby v jeho datových strukturách a ve virtuální síti. Nebyly nalezeny ani žádné chyby, které by způsobovaly pád naší aplikace.

Literatura

[1]

[2] Jiří Kubina. *ARP,TCP,IP utility -zjednodušeně a rychle* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://www.osu.cz/~jura/doc/arp_tcp_ip_utils.pdf>.

[3] Martin Mikulec. *Směrování v síti* [online]. 2010. [cit. 18. 5. 2010]. Dostupné z: <http://owebu.bloger.cz/PC-site/Smerovani-v-siti-routovaci-tabulka-1-dil>.

[4] MICHEK, J. Diplomová práce. In *Emulátor počítačové sítě*, s. 1–78. České vysoké učení technické v Praze, 2008.

[5] Přispěvatelé Wikipedie. *Address Resolution Protocol* [online]. 2010. [cit. 18. 5. 2010]. Dostupné z: <http://cs.wikipedia.org/wiki/Address_Resolution_Protocol>.

[6] Přispěvatelé Wikipedie. *Ethernet* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://cs.wikipedia.org/wiki/Ethernet>.

[7] Přispěvatelé Wikipedie. *Internet Control Message Protocol* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://cs.wikipedia.org/wiki/ICMP>.

[8] Přispěvatelé Wikipedie. *Ifconfig* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://en.wikipedia.org/wiki/Ifconfig>.

[9] Přispěvatelé Wikipedie. *Linková vrstva* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://cs.wikipedia.org/wiki/Spojová_vrstva>.

[10] Přispěvatelé Wikipedie. *Packet tracer* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://cs.wikipedia.org/wiki/Packet_Tracer>.

[11] Přispěvatelé Wikipedie. *Referenční model ISO/OSI* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://cs.wikipedia.org/wiki/Referenční_model_ISO/OSI>.

[12] Přispěvatelé Wikipedie. *Traceroute* [online]. 2010. [cit. 19. 5. 2010]. Dostupné z: <http://cs.wikipedia.org/wiki/Traceroute>.

[13] SOUSEDEK, J. Bakalářská práce. In *Simulátor sítě ve výuce*, s. 1–49. České vysoké učení technické v Praze, 2007.

Dodatek A

Seznam použitých zkratek

2D Two-Dimensional

ABN Abstract Boolean Networks

ASIC Application-Specific Integrated Circuit

⋮

Dodatek B

Instalační a uživatelská příručka

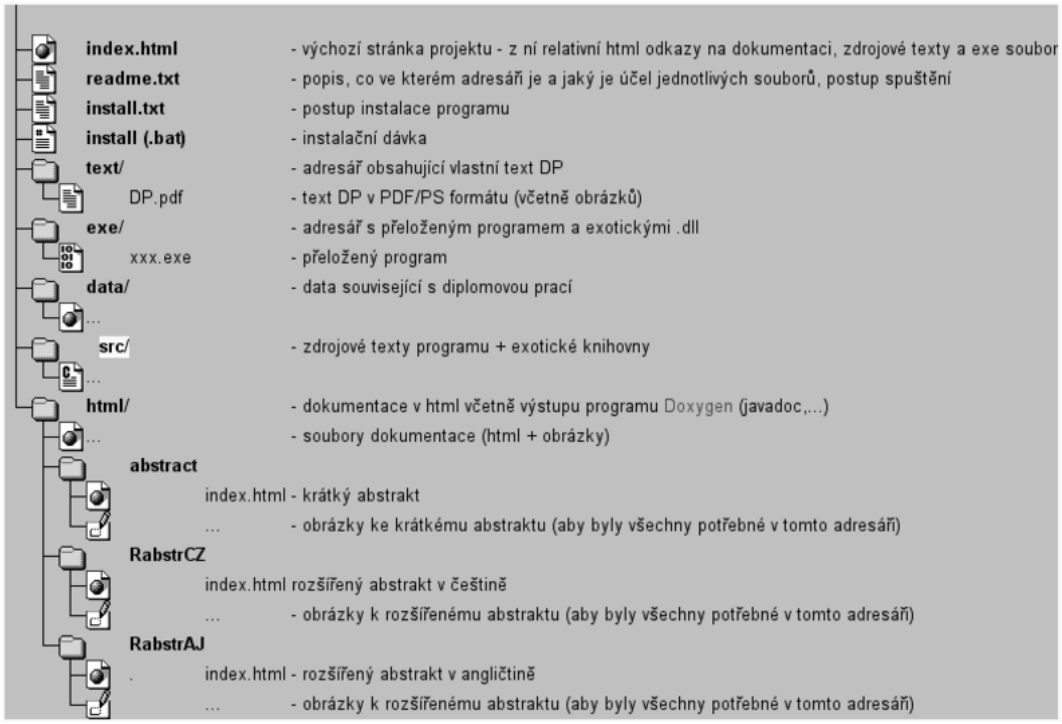
Tato příloha velmi žádoucí zejména u softwarových implementačních prací.

Dodatek C

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 [1]):



Obrázek C.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** (případne 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.