České vysoké učení technické v Praze

Fakulta informačních technologií

Katedra teoretické informatiky

Diplomová práce

Síťový simulátor pro výukové účely na bázi prvků OS Linux

Bc. Tomáš Pitřinec

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

7. května 2012

Poděkování

Děkuji panu Ing. Pavlu Kubalíkovi Ph.D. za dobrý námět k diplomové práci a za pomoc při její realizaci. Děkuji kolegům z vývojového týmu psimulator2 za dobrou spolupráci, bez které by tato práce nemohla vzniknout, všem testerům a korektorům. Také děkuji své rodině za podporu při psaní této práce i během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.
Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 7. května 2012 .....

Odkaz na tuto práci

Tomáš Pitřinec. *Síťový simulátor pro výukové účely na bázi prvků OS Linux: Diplomová práce*. Praha: ČVUT v Praze, Fakulta informačních technologií, 2012.

Abstract

This master thesis inquires into analysis, design, implementation and testing of computer network simulator based on computers with Linux OS. Simulator is intended to be used in subject BI-PSI, Computer networks, on FIT CTU in Prague. The application allows building virtual computer network consisted of computers with OS Linux, configuring the computers in command-line mode, saving current configuration to a given file and loading back. Simulator implements basic versions of following protocols Ethernet II, IP, ARP, ICMP and UDP.

Keywords network simulator, computer networks, network protocols, Linux, router.

Abstrakt

Tato diplomová práce se zabývá analýzou, návrhem, implementací a testováním simulátoru počítačové sítě založené na počítačích s OS Linux. Simulátor má sloužit především pro výukové účely předmětu BI-PSI, Počítačové sítě, na Fakultě informačních technologií ČVUT v Praze. Aplikace umožňuje postavit virtuální počítačovou síť z počítačů s OS Linux, konfigurovat tyto počítače v režimu příkazové řádky, ukládat nakonfigurovanou síť do souboru a znovu ji z něho načítat. V simulátoru jsou implementovány základní verze protokolů Ethernet, IP, ARP, ICMP a UDP.

Klíčová slova síťový simulátor, počítačové sítě, síťové protokoly, Linux, router.

Obsah

Seznam obrázků

Úvod

V předmětu počítačové sítě (BI-PSI) na FIT ČVUT dostávají studenti při laboratorních cvičeních za úkol nakonfigurovat počítačovou síť složenou z počítačů s OS Linux a směrovačů s Cisco IOS. Často však nemají s takovým úkolem žádnou osobní zkušenost, a tak během úlohy řeší různé banální problémy s konfigurací v režimu příkazové řádky a na skutečné síťové experimenty nezbývá čas. Těmto problémům by se mohli studenti vyhnout, kdyby si před samotným cvičením zkusili na simulátoru nakonfigurovat počítačovou síť podobným způsobem, jako v laboratořích. Zároveň by mohli na simulátoru provádět různé pokusy s nastavením počítačových sítí.

V roce 2010 jsme společně se Stanislavem Řehákem jako svůj bakalářský projekt na FEL ČVUT vytvořili Psimulator [1], jednoduchý síťový simulátor pro výukové účely. Ten byl poté nasazen do výuky, kde se osvědčil. Postrádal však grafické uživatelské rozhraní, pro komunikaci s virtuálními počítači nebyl řádně implementován protokol telnet a například linková vrstva byla implementována jen ve velmi omezené míře.

Tato diplomová práce se tedy zabývá novým návrhem a implementací multiplatformního síťového simulátoru psimulator2 jednoduchého na instalaci a použití, který simuluje počítače s operačním systémem Linux nebo Cisco IOS a síťovou komunikaci na linkové, síťové a transportní vrstvě. Moje práce na projektu obnáší především návrh a implementaci jádra simulátoru a implementaci virtuálních počítačů s OS Linux. Na projektu se mnou spolupracovali kolegové Martin Švihlík, který simulátoru vytvářel GUI, Martin Lukáš, který programoval podpůrné komponenty jako připojení po telnetu nebo napojení na grafické rozhraní, a Stanislav Řehák, který se mnou navrhoval a implementoval jádro simulátoru a programoval simulaci síťových prvků Cisco.

Struktura práce

V první kapitole této diplomové práce popisuji specifikaci a cíle projektu Psimulator2, v následující kapitole uvádím některé již existující síťové simulátory a zamýšlím se nad jejich využitím pro výukové účely na FIT ČVUT. Ve třetí kapitole se zabývám analýzou a návrhem simulátoru jako celku. Čtvrtá

[1] jméno vzniklo ze zkratky předmětu PSI a slova simulátor

kapitola podrobně popisuje realizaci jednotlivých částí projektu. Zbývající dvě kapitoly obsahují popis testování aplikace a typy a návody na její další rozšíření.

KAPITOLA 1

Specifikace a cíle projektu

1.1 Popis stávajícího simulátoru

Jak bylo popsáno v úvodu, roku 2010 jsme s kolegou Stanislavem Řehákem v rámci svých bakalářských prací vytvořili síťový simulátor Psimulator. Ten umožňuje simulaci sítě na bázi počítačů s OS Linux a směrovačů s Cisco IOS. Síťové prvky jsou konfigurovány pomocí jednoduchého textového protokolu připomínajícího telnet, telnet samotný však implementován není, a proto se např. napovídání a historie musí řešit externím programem rlwrap. Konfigurace simulátoru se ukládá do XML souboru. Simulátor slouží studentům BI-PSI, má však některé nedostatky, které buď ztěžují jeho použití, nebo brání jeho rozšíření.

Z hlediska použitelnosti má starý simulátor tyto nedostatky:

- Simulátor postrádá grafické uživatelské rozhraní. Studenti, kteří si chtějí postavit vlastní síť, si musí buď napsat nový XML soubor s popisem infrastruktury sítě, nebo musí upravit soubor nějaké stávající sítě.
- Ve starém simulátoru není pořádně implementovaný protokol telnet. Připojí-li se uživatel k virtuálnímu počítači programem telnet, může psát příkazy do jeho příkazové řádky, nefunguje mu však napovídání a historie příkazů. To jsme sice vyřešili použitím programu rlwrap na klientské straně, zachytávání signálů jako `ctrl+C` nebo `ctrl+Z` však tento program již vyřešit neumí. Proto by bylo lepší, kdyby bylo možné se k simulátoru připojit pomocí klasického telnetu.
- Starý simulátor neobsahuje switche, které nebyly v zadání naší bakalářské práce.

Z hlediska návrhu a architektury má simulátor nedostatky, které brání jeho dalšímu rozšíření:

- Ve starém simulátoru neexistují virtuální kabely, počítače jsou spolu propojené pouze pomocí odkazů v síťových rozhraních.
- Všechna činnost simulátoru probíhá ve vláknech komunikace s klientem. Pro každé připojení klienta je totiž v simulátoru vytvořeno vlákno, které pak vykonává vše, co bylo tímto připojením vyvoláno. Je-li tedy např. v nějakém připojení spuštěn program ping, všechny pakety, které jsou v síti v souvislosti s tímto pingem posílány, jsou posílány vláknem tohoto připojení. Tato koncepce zamezuje další rozšiřitelnosti simulátoru, protože broadcastové pakety (např. protokol ARP) by byly mezi počítači posílány sekvenčně a nikoli paralelně.
- Vrstvy ISO/OSI modelu sice byly ve starém simulátoru logicky odděleny, přesto byly prakticky všechny v jedné třídě. Simulace linkové vrstvy byla jen velmi omezená.
- Simulátor podporuje jen jeden typ paketů, ICMP, ve kterém jsou ale informace všech vrstev ISO/OSI modelu.

Starý simulátor vyhovoval požadavkům naší bakalářské práce, ale jeho rozšíření o nové požadavky je neproveditelné, proto jsme se rozhodli vytvořit nový simulátor s tím, že využijeme nejen podstatné části zdrojových kódů starého simulátoru, ale i zkušenosti nabyté při jeho vývoji.

1.2 Cíle projektu

Cílem diplomového projektu Psimulator2 je v programovacím jazyce Java SE navrhnout a implementovat multiplatformní a na instalaci a spuštění jednoduchý síťový simulátor s grafickým uživatelským rozhraním pro potřeby předmětu BI-PSI. Projekt se bude skládat ze dvou samostatných programů, simulačního serveru (psimulator2-backend) a grafického uživatelského rozhraní (psimulator2-frontend). Protože můj podíl na projektu vůbec nezasahuje do frontendu, nebudu se jím dále zabývat, ale budu popisovat pouze psimulator2-backend.

Psimulator2-backend bude mít oproti staré verzi simulátoru podstatně robustnější a modulární architekturu, aby mohl být rozšiřitelný o další moduly, například nové síťové prvky nebo nové síťové moduly. V simulátoru bude možné vytvořit síť virtuálních počítačů, jejíž konfigurace se bude načítat a ukládat do XML souboru. Virtuální síť se bude skládat ze směrovačů a koncových stanic s operačním systémem Linux, směrovačů s Cisco IOS a ze switchů. Tato síťová zařízení spolu budou propojena virtuálními kabely. V simulátoru budou v potřebné míře implementovány protokoly z rodiny TCP/IP, na linkové vrstvě to bude protokol Ethernet, na síťové vrstvě protokoly IPv4 a ICMPv4, protokoly transportní vrstvy budou implementovány pouze omezeně. K virtuálním směrovačům a koncovým stanicím této sítě se

bude možné připojit pomocí protokolu telnet, z pohledu uživatelů se tak bude simulační server tvářit jako skutečná síť. Pokud bude simulátor spuštěn na počítači s OS Linux, bude možno pomocí dalších knihoven a skriptů propojit simulátor s reálnou sítí a to tak, aby bylo možno posílat ping ze simulátoru do reálné sítě a naopak.

Na virtuálních počítačích s OS Linux bude možno konfigurovat pomocí protokolu telnet jejich síťová nastavení. Simulátor 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`. Pomocí dalších příkazů bude možno procházet souborovým systémem počítače a zobrazovat a editovat soubory pomocí jednoduchého textového editoru.

1.3 Požadavky na aplikaci

V následujících odstavcích shrnu konkrétní požadavky na simulátor. Na nový simulátor jsou samozřejmě kladeny i všechny požadavky ze staré verze.

1.3.1 Funkční požadavky

1. Simulátor umožní načíst konfiguraci virtuální sítě ze souboru a zase ji do něho uložit.

2. Simulátor umožní emulovat směrovače Cisco, počítače a směrovače s OS Linux a jednoduché switche.

3. K virtuálním síťovým zařízením, u kterých to má smysl, bude možné se připojit pomocí protokolu telnet, na počítačích s linuxem bude v potřebné míře implementováno rozhraní příkazové řádky.

4. Na virtuálních počítačích s Linuxem budou v potřebné míře implementovány příkazy pro nastavení rozhraní `ifconfig` a `ip addr`.

5. Na virtuálních počítačích s Linuxem budou v potřebné míře implementovány příkazy pro nastavení statického směrování `route` a `ip route`.

6. Na virtuálních počítačích s Linuxem budou v potřebné míře implementovány příkazy pro ověření správnosti konfigurace sítě `ping` a `traceroute`.

7. Virtuální počítač s Linuxem bude mít virtuální souborový systém, budou v potřebné míře implementovány příkazy potřebné k jeho procházení.

8. Pokud simulátor poběží na Linuxu s právy superuživatele, bude možné propojit libovolné síťové zařízení se skutečným počítačem a tak propojit virtuální síť simulátoru se sítí skutečnou, a to tak, aby fungoval příkaz `ping` ze simulátoru do skutečné sítě i naopak.

9. Simulátor bude podporovat statický překlad adres (maškarádu) a v potřebné míře příkaz k jeho nastavení (`iptables -t nat`).

10. Simulátor umožní spuštění déle běžících aplikací (démonů).

11. V simulátoru budou v potřebné míře implementovány tyto protokoly: Ethernet II, ARP, IPv4, ICMP, UDP.

12. Simulátor umožní připojení jiného programu, kterému bude předávat informace o přenášení paketů ve virtuální síti (pro zobrazování paketů v grafickém rozhraní).

1.3.2 Nefunkční požadavky

1. Simulátor bude multiplatformní alespoň pro operační systémy Windows, Linux a Mac OS X.

2. Simulátor bude spustitelný na běžném[2] studentském počítači bez enormních nároků na hardware.

3. Aplikace by měla být co nejvěrnější kopií reálného počítače s Linuxem.

4. Architektura simulátoru bude navržena s ohledem na jeho rozšiřitelnost.

1.4 Rozdělení práce a můj podíl na projektu

Protože se jedná o poměrně rozsáhlý projekt, bude práce na něm rozdělena mezi tým 4 diplomantů. Samotný simulátor se bude skládat ze dvou programů:

- Psimulator2-frontend bude grafické uživatelské rozhraní simulátoru. V jeho editovacím módu si uživatel bude moci vytvořit strukturu virtuální sítě a uložit ji do souboru. V druhém, simulačním, módu se bude moci připojit k již spuštěnému simulačnímu serveru, zachytávat pakety ve virtuální síti a zobrazovat je.

- Psimulator2-backend bude samotný simulační server dle cílů a požadavků popsaných v předchozích odstavcích.

Práce na těchto dvou programech bude mezi členy týmu rozdělena následovně:

1.4.1 Martin Švihlík

Kolega Martin Švihlík má na starost vytvoření celého frontendu.

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

1.4.2 Tomáš Pitřinec (já)

- návrh a implementace jádra simulátoru
- návrh a implementace fyzické vrstvy
- návrh a implementace linkové vrstvy
- implementace částí síťové vrstvy (např. routovací tabulky)
- návrh a implementace aplikací
- linuxový parser příkazů a linuxové příkazy
- analýza, návrh a implementace napojení simulátoru na reálnou síť

1.4.3 Stanislav Řehák

- návrh a implementace jádra simulátoru
- návrh a implementace fyzické vrstvy
- návrh a implementace síťové vrstvy
- návrh a implementace transportní vrstvy
- překlad adres
- návrh aplikací
- aplikace traceroute
- cisco parser příkazů a cisco příkazy
- simulace bufferů na rozhraních, zpoždění

1.4.4 Martin Lukáš

- integrace telnet protokolu
- virtuální souborový systém pro Linux
- načítání a ukládání virtuální sítě do XML souboru
- předávání informací do frontendu
- jednoduchý textový editor pro telnet
- grafický telnet klient ve frontendu

KAPITOLA 2

Existující řešení

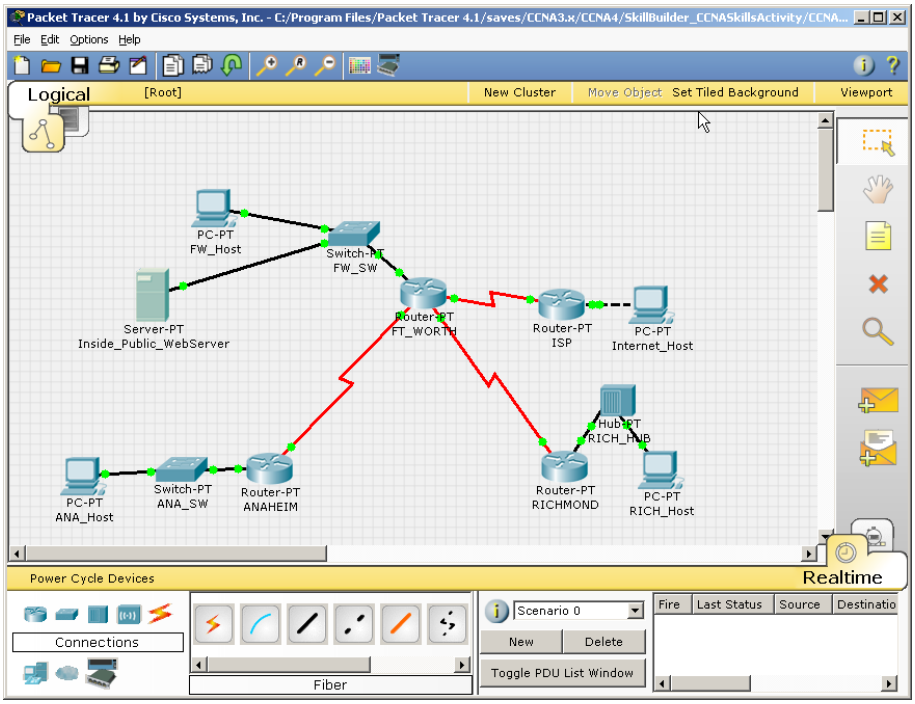
V této části diplomové práce jsem z části vycházel z rešerše své bakalářské práce, která se také zabývá existujícími síťovými simulátory [18]. Síťových simulátorů existuje celá řada a jsou vyvíjeny k různým účelům, například pro výuku nebo pro výzkum a detailní simulaci síťových protokolů. Ne všechny jsou proto vhodné pro výuku předmětu BI-PSI.

2.1 Packet tracer

Nejdříve zmíním snad nejznámější simulační program Packet tracer. Je to simulační software pro výukové účely přímo od společnosti Cisco, který velmi věrně simuluje různé Cisco switche a routery včetně velkého množství síťových protokolů. Má grafické uživatelské rozhraní (obrázek 2.1), zobrazuje i pohyb paketů v síti. Jeho instalace je jednoduchá a použití intuitivní. Lze ho spustit pod Linuxem i pod Windows. Packet tracer je však určen pouze studentům Cisco networking academy, pro které je zdarma [3], ostatním není legálně dostupný. Další jeho velkou nevýhodou je to, že neumožňuje simulovat síťová zařízení s OS Linux.

2.2 NS-2, NS-3

Jiným významným simulátorem je Network simulator. Jedná se o síťový simulátor *„řízený diskrétními událostmi určený především k výzkumu v oblasti počítačových sítí. Podporuje především simulaci TCP, směrovacích a multicastových protokolů v pevných i bezdrátových sítích.“*[11] Původně byl vyvíjen na Univerzitě v Berkeley jako nástupce programu REAL Network Simulator [9], dnes se na jeho vývoji podílí více společností [19]. Celkem byly vydány tři verze tohoto simulátoru: ns-1, ns-2, ns-3, v současnosti se používají



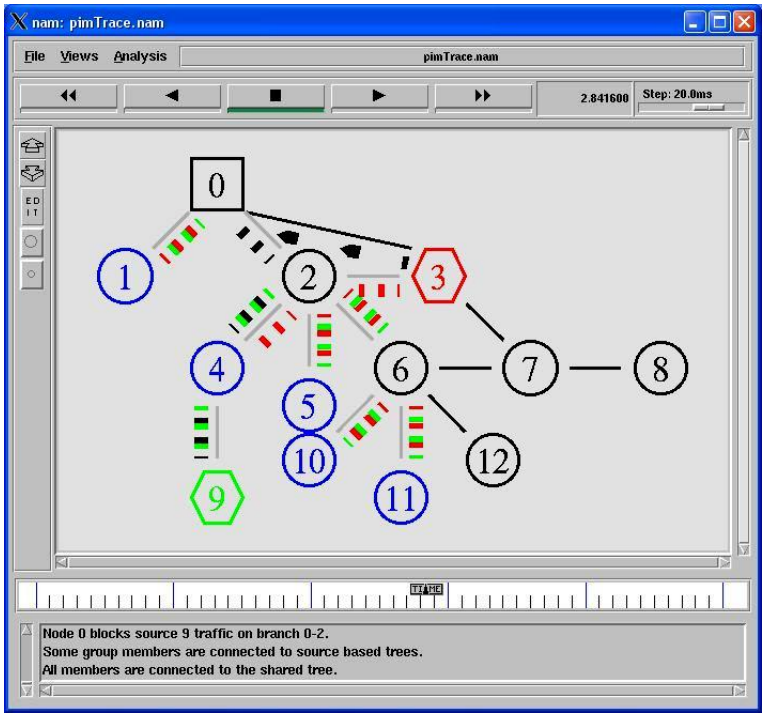
Obrázek 2.1: Packet tracer

především ns-2 a ns-3, který vyšel roku 2008. Předpokládá se, že v budoucnu simulátor ns-3 nahradí ns-2 [19]. Simulátor je volně šiřitelný pod licencí GNU a je dostupný pro unixové systémy, na Windows běží jen přes Cygwin. NS nemá žádné grafické rozhraní pro konfiguraci sítě, síť se konfiguruje pomocí poměrně náročného skriptovacího jazyka a konfigurace sítě zabere poměrně mnoho času [19]. Pro ns-2 existuje nástroj NAM (Network Animator), který umožňuje graficky zobrazit konfiguraci sítě a tok paketů, konfigurovat síť však neumožňuje.

Simulátory ns-2 a ns-3 jsou velmi komplexní a spolehlivé a na mnoha univerzitách slouží k výzkumu síťových technologií. Pro naše účely se však nehodí kvůli celkové náročnosti jejich použití (nutnost učit se skriptovací jazyk), absenci grafického rozhraní a také proto, že neexistuje nativní verze těchto simulátorů pro Windows.

2.3 Simulační framework Omnet++

„Omnet++ je rozšiřitelný a modulární simulační framework napsaný v C++ a určený především k vytváření sítových simulátorů.“ [13]. Díky jeho rozšiřitelnosti je však možné ho použít i k *„simulaci jiných komunikačních systémů, multiprocesorových architektur, testování návrhu hardware, ale dokonce k mo-*



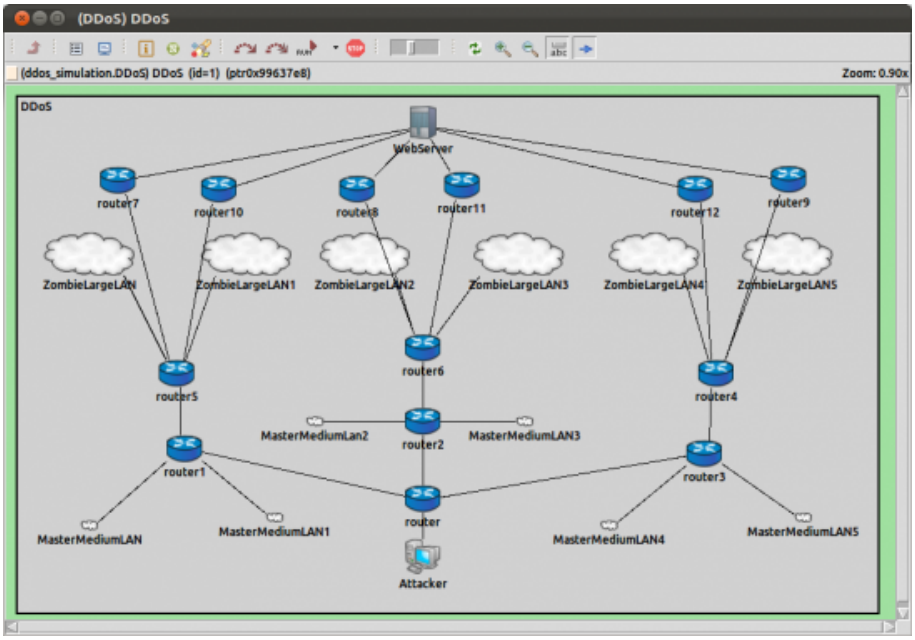
Obrázek 2.2: Network Animator pro ns-2

delování různých obchodních procesů.“ [10]. Simulátor má grafické uživatelské rozhraní, je dostupný na Windows i na Linuxu a pro akademické a nekomerční využití je zdarma. Vyvíjet jej začal roku 1992 András Varga jako svůj studentský projekt na Technické universitě v Budapešti.

Pro Omnet++ existuje mnoho opensource frameworků, které ulehčují konkrétní simulace. Pro síťové simulace je nejznámější projekt INET Framework, který poskytuje sadu protokolů TCP/IP a další síťové protokoly. Dokáže simulovat Cisco IOS i počítač s OS Linux.

Omnet++ je velmi propracovaný a komplexní systém, který se výborně hodí pro simulace síťových protokolů. Pro výukové účely předmětu BI-PSI však není vhodný pro svoji komplexnost a náročnost při instalaci. Student pravděpodobně nebude mít zájem instalovat si tak komplexní simulační systém jen proto, aby si v něm jednou vyzkoušel nakonfigurovat síť o několika počítačích.

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



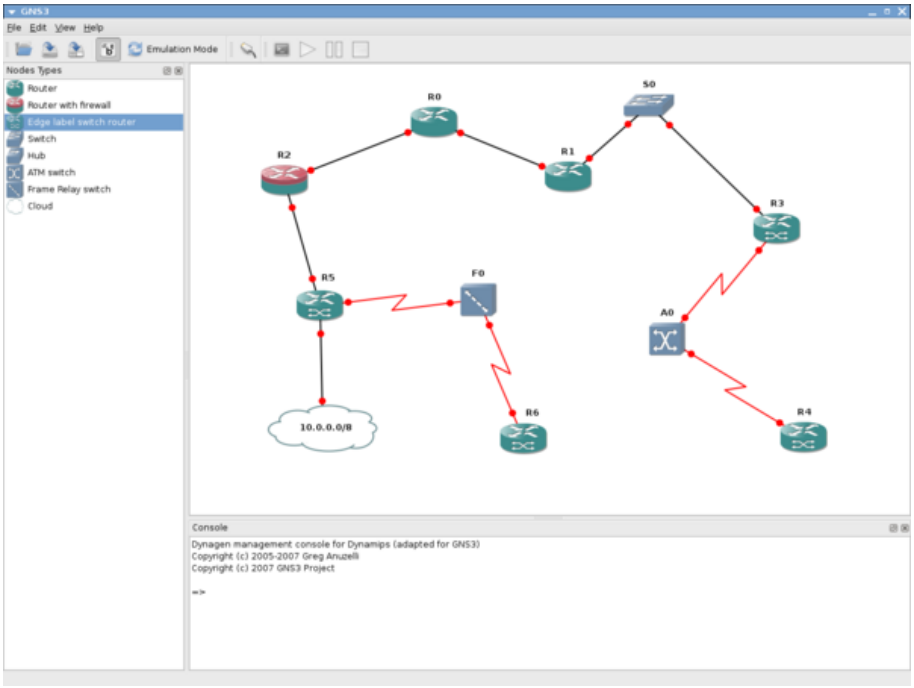
Obrázek 2.3: Simulace v Omnet++ s frameworkem IPNET

2.4 GNS3

GNS3[3] je open-source grafický síťový simulátor [5] dostupný pro Linux, Windows i Mac OS. Je založen na tom, že simulovaná zařízení běží ve virtuálních strojích. Na simulaci Cisco zařízení používá emulátor Dynamips [4], do kterého se musí dohrát IOS, a jeho textový frontend Dynagen. K simulaci počítačů s jinými OS používá buď VirtualBox [4] nebo QEMU [5]. Protože v simulovaných zařízeních běží skutečné operační systémy, umožňuje GNS3 naprosto věrné a komplexní simulace síťových prvků od firmy Cisco i počítačů s OS Linux, Windows nebo Mac. Simulátor umožňuje i napojení do reálné sítě [14].

GNS3 je díky své věrnosti velmi užitečný nástroj pro ty, kteří se více zabývají síťovými technologiemi. Pro naše účely má však tři zásadní nevýhody: Především je nutné mít k dispozici obrazy Cisco IOS, které však mají jen studenti Cisco Networking academy. Druhou nevýhodou simulátoru je jeho poměrně velká náročnost, na jednom průměrném počítači lze simulovat síť o velikosti maximálně 10 - 15 síťových zařízení. Třetí nevýhodou je náročnost jeho instalace, studenti BI-PSI by si kvůli několika málo simulacím museli kromě samotného simulátoru instalovat i zmíněná virtualizační prostředí.

[3] Graphical Network simulator
[4] Virtualbox je open-source multiplatformní virtualizační nástroj
[5] QEMU je open-source multiplatformní emulátor procesoru



Obrázek 2.4: Simulátor GNS3

2.5 NetSim

NetSim je grafický síťový simulátor vyvíjený společností Tetcos určený především pro výukové účely. Protože však jde o proprietární software a protože je dostupný pouze pro Windows, našim účelům nevyhovuje a nebudeme se jím dále zabývat.

2.6 Závěr

Existuje poměrně mnoho dobrých open-source síťových simulátorů, avšak většina z nich buď není multiplatformní, nebo je složitá na instalaci a použití nebo má velké systémové nároky. Studenti PSI potřebují simulátor jednoduchý na instalaci i použití, takový však nebyl nalezen.

KAPITOLA 3

Analýza a návrh

V této kapitole se zabývám analýzou a návrhem aplikace jako celku. Analyzuji požadavky, diskutuji zvolený jazyk a uživatelské rozhraní, navrhuji architekturu aplikace a odhaduji její náročnost. Analýzou a návrhem jednotlivých částí a modulů simulátoru (reálné sítě, linkové vrstvy) se budu zabývat v kapitole o implementaci.

Popisuji již pouze backend simulátoru, tak jak byl vymezen v 1.2.

3.1 Analýza požadavků

3.1.1 Příkazová řádka na Linuxu

Kompletní implementace bashe nebo příkazů jako `ifconfig` nebo `route` by daleko překračovala rozsah této diplomové práce. Přesto by příkazová řádka i jednotlivé příkazy měly být implementovány tak, aby na první pohled vypadaly jako ve skutečné příkazové řádce Linuxu a aby byly podporovány všechny v dané situaci obvyklé varianty příkazů. Příkazová řádka by tedy měla podporovat vypisování promptu včetně aktuální cesty, signály jako `ctrl+C`, napovídání nebo historii příkazů tak, jak funguje na skutečném linuxovém počítači. Z příkazů jako `ifconfig`, `route`, `ip addr` nebo `ip route` by měly být implementovány ty části, které studenti běžně používají. Například příkaz `ifconfig` by měl podporovat změnu IP adresy a masky na rozhraní, nemusí však podporovat například změnu mac adresy nebo přidání více IP adres na jedno rozhraní.

3.1.2 Podpora síťových protokolů

V simulátoru mají být v potřebné míře implementovány protokoly Ethernet II, ARP, IPv4, ICMP a UDP. Jejich kompletní implementace by však přesahovala rozsah této práce, proto budou implementovány pouze v míře potřebné

k simulaci. Nebudu se například zabývat položkami v hlavičkách paketů, které by byly v simulátoru zbytečné.

3.1.3 Napojení na reálnou síť

Napojení virtuální sítě simulátoru na reálnou síť není pro výukové účely nutné, půjde však o zajímavou vlastnost, která uživateli umožní „dopingnout“ se z virtuálního počítače na svůj skutečný počítač nebo na nějaký server v internetu. Ke spojení virtuální sítě s reálnou sítí bude třeba konfigurovat i hostitelský počítač, na kterém bude simulátor spuštěn. K tomu bude uživatel pravděpodobně potřebovat superuživatelské oprávnění. Více se analýzou tohoto požadavku budeme zabývat v části 4.10.

3.2 Programovací jazyk a uživatelské rozhraní

3.2.1 Programovací jazyk

Simulátor jsme se rozhodli programovat v jazyce Java z několika důvodů. Java poskytuje velký programátorský komfort, stabilitu a zároveň možnost spouštět aplikaci pod různými operačními systémy, což nám zajistí splnění jednoho z nefunkčních požadavků. Java je poměrně používaným programovacím jazykem, a proto pro ni existuje velké množství volně dostupných knihoven, které můžeme v naší práci využít.

3.2.2 Vývojové prostředí

Práci budu implementovat ve vývojovém prostředí NetBeans[6]. Ke sdílení a verzování kódu budeme používat systém subversion[7] na serveru Google Code[8].

3.2.3 Uživatelské rozhraní

Grafické uživatelské rozhraní simulátoru je obsaženo v jeho frontendu, backend proto bude spouštěn jednoduše v příkazové řádce. Na konzoli bude případně vypisovat různá hlášení. K virtuálním počítačům se uživatel bude připojovat buď pomocí frontendu, nebo programem telnet dostupným prakticky na kterékoliv platformě.

3.3 Analýza a návrh architektury

Architekturu nového simulátoru navrhujeme společně se Stanislavem Řehákem. Velmi nám pomohlo, že oba již máme zkušenosti s návrhem a implementací

[6] http://netbeans.org/
[7] http://subversion.apache.org/
[8] stránka projektu https://code.google.com/p/psimulator/

starého simulátoru. Návrh architektury ve dvojici se osvědčil, protože se jedná o velmi složitý systém, ve kterém by jednotlivec mohl přehlédnout důležité detaily. Diskuzí návrhu jsme toto riziko podstatně snížili.

3.3.1 Architektura jádra

Jádro simulátoru bude tvořeno na sobě nezávislými virtuálními počítači, síťovými zařízeními a kabely stejně, jako je skutečná síť tvořena jednotlivými počítači a spojena kabely. Při startu simulátoru se dle konfiguračního souboru vytvoří, pospojují a spustí virtuální počítače. V simulátoru poběží telnet server, který bude dle zadaného portu připojovat virtuální počítače. Zařízení ve virtuální síti si budou posílat pakety po virtuálních kabelech.

3.3.2 Virtuální síťové zařízení

Virtuální síťové zařízení je základní jednotkou simulované virtuální sítě. Podporovány budou směrovače s OS Linux, směrovače Cisco a jednoduché switche. Obecně však musí být zařízení navrženo tak, aby bylo v budoucnu možné přidat nové síťové zařízení (např. hub, bridge) nebo změnit a rozšířit stávající. Každé síťové zařízení bude mít svůj **fyzický modul**, který bude simulovat veškerý síťový hardware a posílat a přijímat pakety. **Síťový modul** virtuálního síťového zařízení se bude starat o veškerou síťovou komunikaci: směrování, filtrování, posílání a odpovídání. Vzhledem k tomu, že v simulátoru je několik typů síťových zařízení, bude muset být i více typů síťových modulů. Směrovače s Linuxem musí mít svůj virtuální **souborový systém**, **parser příkazů** a možnost spustit různé **síťové aplikace**. Aplikace musí být navrženy tak, aby bylo v budoucnu možné do simulátoru doprogramovat například jednoduchý DHCP server, HTTP server, DNS server apod.

3.3.3 Fyzický modul

Fyzický modul virtuálního počítače bude simulovat veškerý síťový hardware zařízení. Při odesílání bude přijímat pakety od síťového modelu a posílat je po kabelech. Pakety přijaté od kabelů bude předávat síťovému modulu. Zdířka k zapojení kabelu je u všech síťových zařízení nazývána switchport. K poslání paketu dostane modul zároveň informaci, přes který switchport má být paket odeslán, při příjmu paketu předá síťovému modulu zároveň s paketem i číslo switchportu, na který paket přišel.

3.3.4 Síťový modul

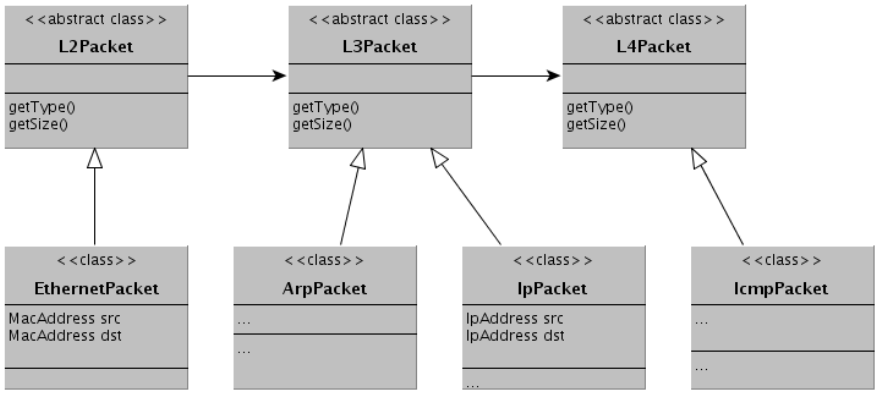
Síťový modul virtuálního počítače se bude starat o veškerou síťovou komunikaci síťového zařízení. Tuto činnost rozdělíme dle vrstev ISO/OSI modelu, síťový modul se tedy bude skládat z linkové, síťové a transportní vrstvy.

Přijatý paket bude nejprve zpracován linkovou vrstvou, bude-li vše v pořádku, předá ho linková vrstva síťové vrstvě. Síťová vrstva paket buď přepošle dále, nebo ho předá transportní vrstvě, podle toho, zda je určen pro tento počítač nebo pro jiný. Transportní vrstva na přijatý paket buď sama odpoví, nebo ho předá nějaké aplikaci, je-li paket určen jí. Aplikace při odeslání předá paket transportní vrstvě, ta síťové a ta pak linkové, přičemž každá z vrstev přidá paketu svoji hlavičku.

Bude implementováno více síťových modulů pro různé typy síťových zařízení. Síťový modul switche bude obsahovat pouze linkovou vrstvu, modul routeru bude obsahovat všechny vrstvy. Je třeba navrhnout modul abstraktně, aby mohl být v budoucnu do simulátoru přidán úplně nový modul.

3.3.5 Síťová komunikace

Počítače si budou mezi sebou posílat pakety, které budou v simulátoru reprezentovány objekty různých typů. Oproti starému simulátoru, kde byla pro všechny pakety jedna třída, jsme se rozhodli rozčlenit pakety dle vrstev ISO/OSI modelu. Paket linkové vrstvy (správně rámec) bude mít jako data paket 3. vrstvy, ten bude mít jako data datagram 4. vrstvy a ten případně nějaká data vrstvy aplikační. Zřetězení paketů je vidět na obrázku 3.1. Pakety budou předávány voláním metod virtuálních počítačů.



Obrázek 3.1: Návrh struktury paketů.

Pakety jsou během své cesty často měněny, mění se např. ttl v IP hlavičce nebo `dstmac` a `srcmac` v hlavičce Ethernetu. Přitom je třeba pakety logovat, vypisovat a předávat frontendu simulátoru. Proto budou implementovány tak, aby se jednou vytvořený paket již nedal změnit. To zamezí tomu, aby se zalogoval nějaký paket v určité podobě, ale pak se odeslal frontendu v podobě mezitím změněné.

3.3.6 Vlákna

Ve starém simulátoru běželo každé telnetové spojení s uživatelem v samostatném vlákně, veškerá činnost vyvolaná určitým spojením byla vykonána jeho vláknem. Posílání paketů bylo realizováno voláním metod, které si mezi sebou paket předávaly. To však znamená, že příkaz `ping` a veškerá síťová komunikace na různých počítačích s ním spojená byla vykonávána ve vlákně shellu, ve kterém byl příkaz spuštěn. Toto řešení stačilo ve staré verzi simulátoru, avšak ukázalo se jako značně omezující pro jeho další rozšiřování. Například při posílání paketů na broadcast by byly pakety na počítačích zpracovávány sekvenčně, teprve po zpracování paketů na jednom počítači by mohl být paket poslán a zpracován na dalším počítači. Také by nebylo možné čekat na vypršení timeoutu.

Ve skutečném počítači běží každá úloha nebo proces ve vlastním vlákně, z výše popsaných důvodů tomu tak bude i v našem simulátoru. Ve svém vlastním vlákně poběží:

- každý fyzický modul virtuálního počítače
- každý kabel
- linková vrstva síťového modulu virtuálního počítače
- síťová vrstva síťového modulu
- každý shell
- každá spuštěná aplikace

To je vzhledem k možnému počtu simulovaných počítačů poměrně velké množství vláken na jednu aplikaci, proto bude třeba, aby se vlákna uspávala v době, kdy nevykonávají žádnou užitečnou práci. Tímto problémem se budu více zabývat v jejich realizaci. Vlákna si mezi sebou budou předávat data, většinou pakety, pomocí bufferů a zároveň s tím se budou navzájem probouzet.

3.4 Odhad náročnosti

Backend simulátoru nebude mít žádné grafické rozhraní, nebude přistupovat do žádné databáze a její datové struktury budou poměrně jednoduché. Proto by simulace virtuální sítě o velikosti kolem dvaceti počítačů včetně načteného prostředí JRE neměla zabrat více jak 100 MB operační paměti. Simulátor nebude provádět žádné náročné operace. Jediné, co by mohlo zatěžovat procesor, je počet vláken běžících v simulátoru. Protože se však vlákna budou uspávat a probouzet pouze na vykonání celkem nenáročné operace, měla by být procesorová náročnost zanedbatelná.

3.5 Odhad náročnosti implementace

Rozsah implementace backendu odhadujeme na 20 000 řádků, simulátor by měl být implementován do konce března 2012.

KAPITOLA 4

Realizace

V této kapitole se zabývám realizací celého simulátoru a především těch částí, které jsem implementoval já. Nejprve popisuji realizaci různých podpůrných částí simulátoru, jako jsou uspávající se vlákna nebo logování, a poté implementaci samotné virtuální sítě.

4.1 Vlákna

Z analýzy aplikace vyplynulo, že většina komponent virtuální sítě poběží ve vlastním vlákně. Přitom však tato vlákna nebudou mít po většinu času simulace prakticky nic na práci. Jejich práce spočívá pouze v tom, že vezmou data z nějakého bufferu, něco výpočetně nenáročného s nimi provedou a uloží je do jiného bufferu. Samozřejmě by bylo možné, aby každé vlákno ve věčné smyčce kontrolovalo své buffery, jestli v nich náhodou nejsou data ke zpracování, takové řešení by však bylo neúnosně náročné. Řešením tohoto problému je tzv. uspávací vlákno, které samo přestane pracovat a uspí se, pokud nemá práci, a které se nechá probudit jiným vláknem, pokud práci dostane.

4.1.1 Implementace uspávacího vlákna

Toto uspávací vlákno je v simulátoru implementováno třídou `WorkerThread`. Modul, který ho chce využívat, musí implementovat rozhraní `SmartRunnable`. Jejich strukturu můžeme vidět na obrázku 4.1. Rozhraní `SmartRunnable` deklaruje metodu `doMyWork()`, která má obsahovat všechnu činnost daného modulu. `WorkerThread` spouští své vlákno v Javě naprosto standardním způsobem. Sám implementuje rozhraní `Runnable` a vlastní instanci třídy `Thread` myThread. Vlákno spouští standardně voláním `myThread.start()`, které spustí jeho metodu `run()` v novém vlákně. Ta je implementována přibližně takto (pro přehlednost jsem smazal nedůležité řádky):

```
@Override
public void run() {
	while (true) {
		smartRunnable.doMyWork();	// vykonani pridelene prace
		synchronized(this){
			isSleeping = true;
			wait(); // prepnuti vlakna do stavu wait
		}
	}
}
```

Metoda `run()` tedy ve věčné smyčce nejprve vykoná přidělenou práci voláním `smartRunnable.doMyWork()` a potom se její vlákno zavoláním `wait()` přepne ze stavu `running` do stavu `wait`[9] a tak se spustí. Má-li být probuzeno, zavolá jiné vlákno metodu `wake()`, která je implementována přibližně takto:

```
public synchronized void wake() {
	if (isSleeping) {	//kdyz nebezi tak se zapne
		isSleeping = false;
		this.notifyAll();	// probudi vlakno
	}
}
```

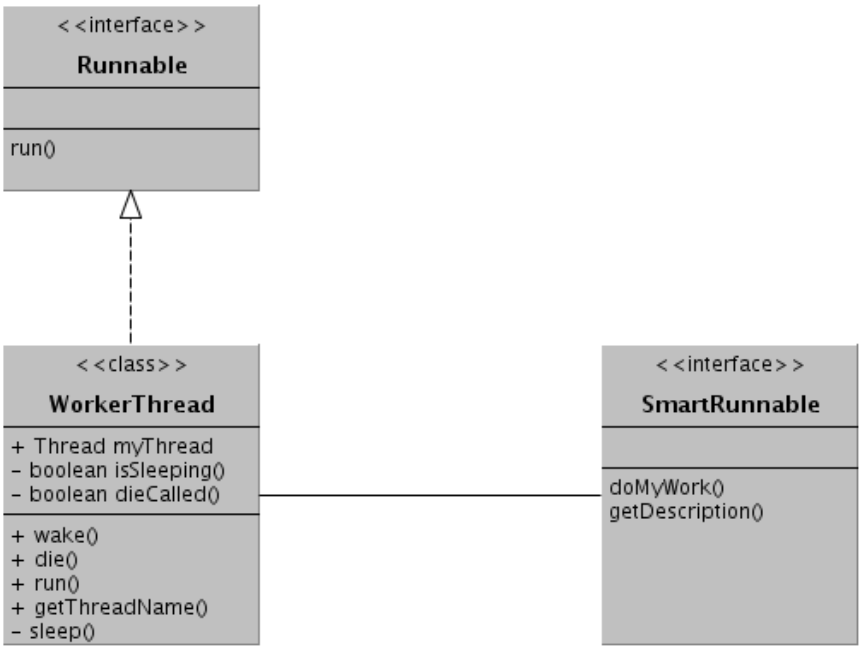
Voláním `notifyAll()` se vlákno přepne znovu do stavu `lock` a tak se spustí.

Před započetím implementace simulátoru jsme nejprve naprogramovali prototyp tohoto uspávajícího vlákna a ten jsme vyzkoušeli. Zjistili jsme, že tento způsob procesor nezatěžuje a proto jsme ho v simulátoru implementovali. Pro větší přehlednost při debugování je vlákno v simulátoru pojmenované. Aby mohly být aplikace využívající toto vlákno spouštěny a ukončovány, má vlákno metodu `die()`, která ho definitivně ukončí.

4.1.2 Typické použití

Typické použití uspávacího vlákna je například v modulu linkové vrstvy. Linková vrstva implementuje rozhraní `SmartRunnable` a `WorkterThread` si nastartuje ve svém konstruktoru. V metodě `doMyWork()` prochází své buffery, jestli v nich není paket k vyřízení. Pokud jsou buffery prázdné, uspí se. Pokud jí nějaké jiné vlákno do bufferů přidá nějaký paket (u linkové vrstvy je to buď fyický modul nebo síťová vrstva), zavolá již samotným přidáním metodu `wake()` a vlákno linkové vrstvy začne vykonávat svoji práci, tedy kontrolovat buffery a předávat pakety.

[9] více o stavech vláken v Javě v [7]



Obrázek 4.1: Třídní diagram uspávacích vláken

4.1.3 Budík

Uspaná vlákna v simulátoru jsou obvykle buzena nějakou událostí, např. příchodem paketu. Občas je ale potřeba, aby se po uplynutí nějaké doby (vypršení timeoutu) sama od sebe vzbudila a vykonala nějakou činnost. K tomuto účelu jsem v simulátoru ve třídě `Alarm` implementoval tzv. budík. Při simulaci běží v systému jeden budík dostupný všem objektům, které si u něho mohou zaregistrovat čas probuzení. Podmínkou je, aby klient budíku implementoval rozhraní `Wakeable`, jehož metodou `wake()` je objekt probuzen.

4.2 Logování

4.2.1 Požadavky na logování

Logování v simulátoru neslouží pouze k vypisování zpráv na standardní výstup nebo do nějakého souboru, ale stará se o zachytávání a zpracování všech událostí, které se v simulátoru dějí. Byly na něj kladeny tyto požadavky:

- Logování bude umět klasicky vypisovat zprávy na standardní výstup nebo do souboru. U každé zprávy bude informace, kterou částí systému byla zalogována.

- Bude možné logovat zprávy s různou úrovní důležitosti.
- Bude možné logovat zprávy různých kategorií podle toho, které části systému se týkají.
- Na standardní výstup bude možné filtrovat zprávy dle jejich důležitosti a kategorie.
- Loggeru bude možné předat kromě samotné zprávy i objekt, který loguje, a případně nějaký další objekt, kterého se zpráva týká (např. posílaný paket nebo vyhozenou výjimku).
- Logger bude umožňovat předávat zprávy jiným částem systému, například frontendu bude předávat informace o toku paketů.

4.2.2 Implementace

4.2.2.1 Struktura logovacího systému

Logovací systém byl navržen tak, aby každou zprávu, kterou dostane, mohl předat více různým objektům. Logger je implementován ve statické třídě `Logger`. Ten sám zprávy nijak nezpracovává, pouze je předává listenerům, které jsou u něho zaregistrované. Listenery implementují rozhraní `LoggerListener` s metodou listen(...), pomocí níž logger předává zprávy listenerům. Listenerem může být v podstatě jakákoliv třída, zatím je implementován `SystemListener`, který se stará o vypisování na standardní výstup, a `PacketTranslator`, který frontendu posílá informace o paketech v síti. V budoucnu se předpokládá využití listeneru např. pro příkaz `debug` na Ciscu nebo `tcpdump` na Linuxu.

4.2.2.2 Logovací zprávy

Každá logovací zpráva obsahuje kromě samotného textu i úroveň důležitosti (loglevel), logovací kategorii a odkaz na objekt, který zprávu zalogoval. Může obsahovat ještě další libovolný objekt, který se nějak ke zprávě vztahuje. Například při zprávě o odeslání paketu je možné zalogovat i odeslaný paket. Všechny tyto informace se předávají listenerům. V simulátoru máme tyto loglevely:

1. **error**: závažná chyba programu, na standardní výstup se vypíše hlášení a program se ukončí
2. **warning**: chyba v systému, ale simulace může pokračovat
3. **important**: důležité informace, které je vždy potřeba vypsat na standardní výstup, například uložení konfigurace do souboru
4. **info**: běžné informace o událostech v síti
5. **debug**: ladící hlášení

4.2.2.3 SystemListener

`SystemListener` vypisuje události na standardní výstup. Na začátku běhu simulátoru se ve třídě `ConfigureSystemListener` nakonfiguruje, jaké zprávy se mají vypisovat. Pro danou kategorii se vypisují všechna hlášení se stejnou nebo vyšší úrovní důležitosti, než jaká byla nakonfigurována. Implicitně je pro všechny kategorie nastaven loglevel **important**, to znamená, že se vypisují i **warning** a **error**. Přidávám příklad zalogovaného hlášení o tom, že bylo nastaveno napojení do reálné sítě:

```
[IMPORTANT] REAL_NETWORK: router2: RealSwitchport 1: Real interface
sim0 tied together with switchport n. 1. Connection to real network
started.
```

4.3 Virtuální síťové zařízení

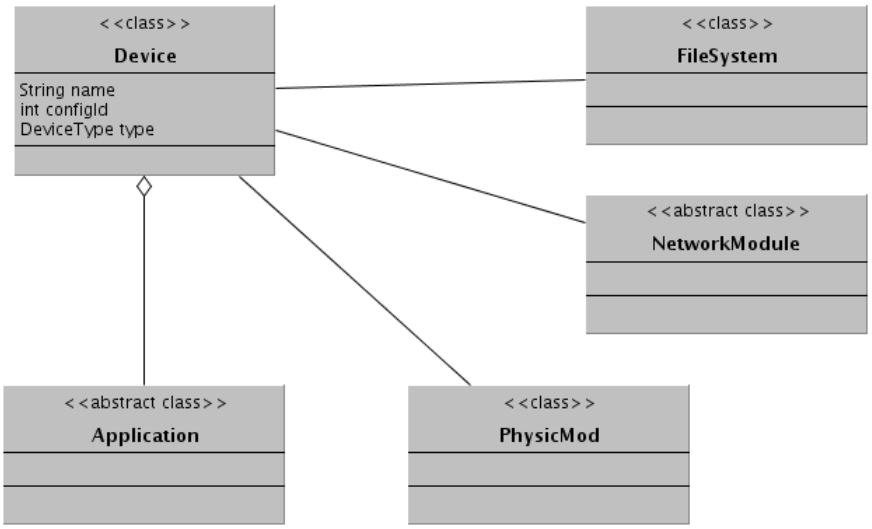
Virtuální síťové zařízení implementované ve třídě `Device` je realizací všech druhů síťových zařízení simulované virtuální sítě. Zařízení se skládá z těchto komponent:

- **Fyzický modul** simuluje fyzická síťová zařízení počítače.
- **Síťový modul** simuluje část jádra OS, která má na starosti síťovou komunikaci.
- **Shell a parser příkazů** slouží k připojení se na zařízení.
- Na zařízení je možné spustit **aplikace**.
- **Filesystém** simuluje reálný filesystém zařízení.

Zatím jsou v simulátoru implementované tyto typy zařízení:

- `cisco_router` reprezentuje router s Cisco IOS.
- `linux_computer` reprezentuje router, notebook nebo PC s operačním systémem Linux. Jednotlivé podtypy se v backendu liší pouze počtem rozhraní.
- `simple_switch` reprezentuje jednoduchý switch. Na něj není možno se připojit telnetem, spouštět příkazy nebo aplikace ani procházet filesystém.

Ve staré verzi simulátoru bylo síťové zařízení implementováno v abstraktní třídě `AbstraktniPocitac` a jednotlivé typy od něho dědily. To v nové verzi již není potřeba, protože rozdílné chování jednotlivých typů zařízení je implementováno v jeho komponentách. Linuxový počítač a switch jsou tedy v simulátoru objekty stejného typu, liší se však typem síťového modulu a tím,



Obrázek 4.2: Třídní diagram síťového zařízení

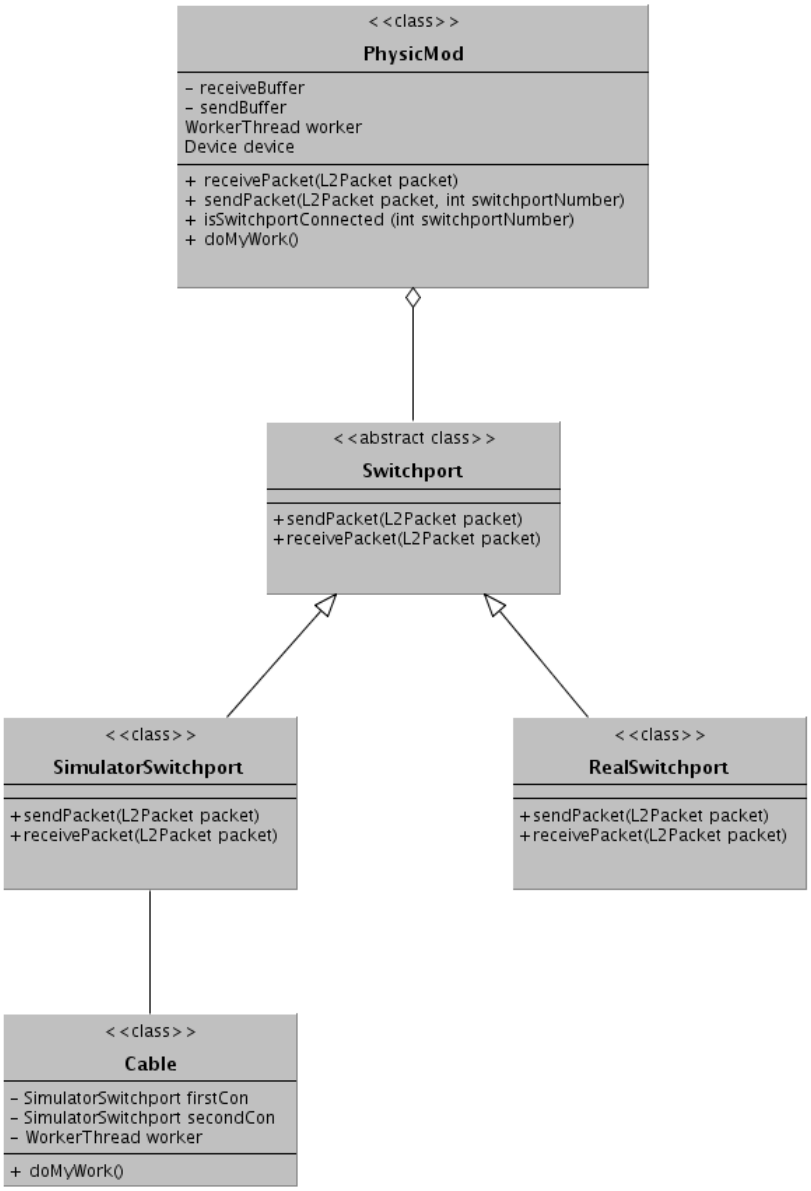
že na switch není možné se připojit pomocí telnetu. V budoucnu je možné přidat nový typ zařízení, který může různě kombinovat již implementované nebo i nové typy komponent.

4.4 Fyzický modul

Fyzickým modulem se zde zabývám jen stručně, protože většinu implementace měl na starosti kolega Stanislav Řehák; navrhovali jsme ho však dohromady.

4.4.1 Struktura modulu

Modul simuluje fyzická zařízení počítače, která mají na starost síťovou komunikaci. Celý modul běží ve vlastním vlákně. Je implementován ve třídě `PhysicMod`, třída `Switchport` reprezentuje jednu zásuvku na síťový kabel a to u switche, routeru i počítače. V simulátoru jsou dva druhy těchto zásuvek. `SimulatorSwitchport` reprezentuje virtuální switchport uvnitř systému, lze do něj zapojit jeden virtuální kabel, který je v simulátoru reprezentován třídou `Cable`. `RealSwitchport` je reprezentací switchportu, který lze svázat se skutečným rozhraním hostitelského počítače a napojit tak simulátor na reálnou síť. Více se jím budu zabývat v části o napojení na reálnou síť (4.10).



Obrázek 4.3: Struktura fyzického modulu

4.4.2 Rozhraní k jiným částem systému

Fyzický modul byl navržen maximálně nezávisle na ostatních částech systému. Jeho rozhraní směrem k síťovému zařízení (a hlavně tedy k jeho síťovému modulu) tvoří pouze dvě metody třídy `PhysicMod`: `sendPacket` a `isSwitchportConnected`. První pošle paket na zadaný switchport a druhá

zjistí, je-li zadaný switchport po kabelu spojen s jiným switchportem. Přijaté pakety jsou předávány síťovému modulu jeho metodou `receivePacket`.

4.4.3 Popis činnosti

Kabely pomocí switchportů předávají pakety fyzickému modulu počítače do jeho bufferu. Stejně tak i síťový modul síťového zařízení předává fyzickému modulu pakety k odeslání do jiného bufferu. Veškerá činnost fyzického modulu pak spočívá v tom, že si kontroluje tyto dva buffery, pakety z nich vybírá a předává na místo jejich určení.

4.5 Síťový modul

Síťový modul jsme s kolegou Řehákem společně navrhovali a jeho implementaci jsme si rozdělili. Kolega implementoval síťovou a transportní vrstvu, já jsem implementoval linkovou vrstvu, kterou popíši v následující části. Zde se zabývám implementací síťového modulu jako celku.

4.5.1 Funkce síťového modulu

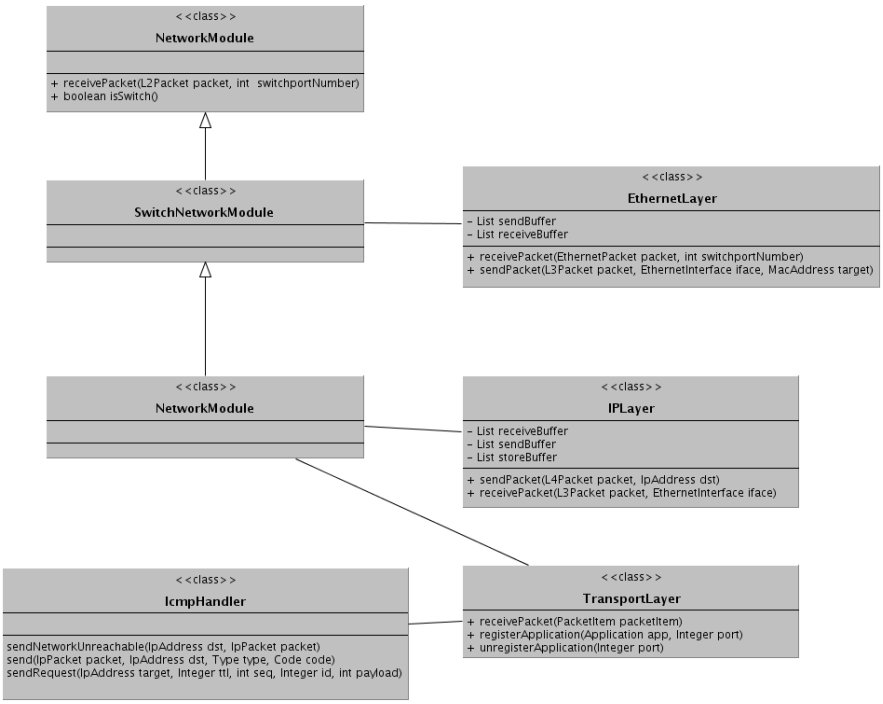
Síťový modul virtuálního síťového zařízení se stará o veškerou síťovou komunikaci zařízení na úrovni softwaru. Vykonává tedy část práce, kterou na skutečném počítači vykonává jádro operačního systému. V simulátoru existuje několik typů síťového modulu pro různá síťová zařízení. `SwitchNetworkModule` je síťový modul pro switch, který zajišťuje směrování rámců na linkové vrstvě. `IpNetworkModule` je síťový modul pro síťové zařízení pracující na síťové vrstvě, zajišťuje přijímání, odesílání a přeposílání paketů na 3. vrstvě ISO/OSI modelu, adresaci, směrování a překlad adres.

4.5.2 Struktura síťového modulu

Síťový modul je rozdělen dle jednotlivých vrstev ISO/OSI modelu. To umožňuje přehlednou a jednoduchou implementaci jednotlivých vrstev, ale i možnost jednoduše rozšířit některou z vrstev modulu. Modul pro switch obsahuje pouze linkovou vrstvu, modul pro router se skládá z linkové, síťové a transportní vrstvy.

4.5.2.1 Linková vrstva

Linková vrstva implementovaná ve třídě `EthernetLayer` se stará o přijímání a posílání rámců na 2. vrstvě ISO/OSI modelu. Implementován je protokol Ethernet II. Vrstva běží ve vlastním vlákně. Její implementací se zabývám v části 4.7.



Obrázek 4.4: Struktura síťového modulu

4.5.2.2 Síťová vrstva

Síťová vrstva je implementována ve třídách balíčku `networkModule.L3`. Podporuje protokoly IP a ARP pro zjišťování fyzických adres. Vrstva běží ve vlastním vlákně. Implementací této vrstvy se ve své diplomové práci zabývá kolega Stanislav Řehák [23]. Routovací tabulka byla převzata ze starého simulátoru, tou se zabývám ve své bakalářské práci já [18].

4.5.2.3 Transportní vrstva

V našem simulátoru nedochází k žádnému přenosu užitečných dat, transportní vrstva je proto implementována jen omezeně pro protokoly UDP a ICMP. Protokol ICMP bývá sice řazen do síťové vrstvy ISO/OSI modelu [12], v simulátoru je však až na transportní, protože jeho datagramy jsou přenášeny uvnitř IP paketů. Vrstva tvoří rozhraní mezi síťovým modulem virtuálního počítače a síťovými aplikacemi, které na nějakém portu přijímají a odesílají data. Z tohoto důvodu si u ní mohou síťové aplikace zaregistrovat port, na kterém poslouchají. Všechny pakety přicházející na tento port pak transportní vrstva předává zaregistrované aplikaci. Vrstva nemá vlastní vlákno, její činnost je tedy vykonávána ve vlákně síťové vrstvy nebo ve vlákně aplikací, podle toho,

kdo její metody volá.

4.5.3 Činnost síťového modulu

Činnost síťového modulu popisuji na příkladu odeslání `ICMP request` a přijetí `ICMP reply`. Paket je odeslán aplikací ping[10], která zavolá příslušnou metodu `IcmpHandleru` (viz obrázek 4.4). `IcmpHandler` vytvoří ICMP datagram s požadovanými parametry, předá ho síťové vrstvě (`IPLayer`) do bufferu paketů k odeslání a probudí její vlákno. `IPLayer` vyjme paket z bufferu, přidá k němu IP hlavičku a snaží se paket předat linkové vrstvě. Pokud zná fyzickou adresu cíle, předá paket linkové vrstvě rovnou, pokud ne, zeptá se po síti pomocí protokolu ARP, mezitím si paket uloží do svého `storeBufferu`. Jakmile zná cílovou mac adresu, předá paket i s adresou do bufferu linkové vrstvě a probudí její vlákno. Ta ho z bufferu vyjme, přidá ethernetovou hlavičku a předá paket do bufferu fyzického modulu, čímž je paket pro síťový modul odeslaný.

Při přijímání paketu dostane paket do bufferu nejprve linková vrstva (`EthernetLayer`). Pokud je paket určen tomuto počítači, předá ho linková vrstva již bez ethernetové hlavičky síťové vrstvě. Je-li paket i na síťové vrstvě určen tomuto počítači, předá ho síťová vrstva transportní vrstvě. V případě ICMP paketu pak `IcmpHandler` předá paket příslušné aplikaci. U protokolu TCP nebo UDP se cílová aplikace určuje pomocí portu, u ICMP pomocí identifikátoru.

4.5.4 Rozhraní k jiným částem systému

Rozhraní k fyzickému modulu tvoří pouze metoda síťového `receivePacket`, síťový modul pro odesílání paketů využívá metody `sendPacket` fyzického modulu. Rozhraní k síťovým aplikacím je obsáhlejší, aplikace se registrují přímo transportní vrstvě a přes ní přijímají a odesílají pakety. Většina parametrů síťového modulu je konfigurována pomocí příkazů jako `ifconfig`, `route` atd. Tyto příkazy přistupují přímo k vnitřním strukturám síťového modulu, což je však v souladu s návrhem, protože tyto příkazy jsou k tomu přímo určeny.

4.5.5 IP adresa

4.5.5.1 Analýza a požadavky

IP adresa je velmi důležitou datovou strukturou nejen síťového modulu, proto se jí zde zabývám poněkud podrobněji. Ve staré verzi simulátoru adresa obsahovala kromě samotné IP adresy také svoji masku, aby bylo možno použít ji jako parametr rozhraní, a port, aby bylo možno ji použít v paketech, které tehdy neměly oddělené hlavičky. Toto řešení se však ukázalo jako nevyhovující, protože v každém jejím využití byl buď parametr port, nebo parametr maska

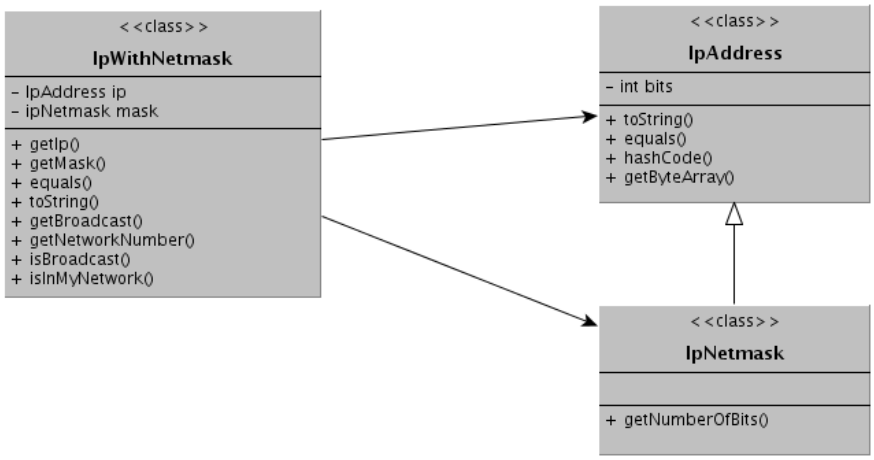
[10] O té více v práci kolegy Řeháka [23]

nesmyslný. Proto jsem se rozhodl udělat pro IP adresu nové datové struktury, které zachovávají stávající funkčnost. Na nové datové struktury byly kladeny tyto především požadavky:

- IP adresu bude možno vytvořit z řetězce, adresu bude možno vypsat jako řetězec
- IP adresa a maska bude po svém vytvoření neměnná
- IP adresu bude možno porovnat s jinou IP adresou
- IP adresa s maskou bude umět vypočíst číslo své sítě a broadcast

Protože jsem nenašel žádnou již implementovanou datovou strukturu, která by těmto požadavkům vyhovovala, implementoval jsem ji sám.

4.5.5.2 Struktura a implementace



Obrázek 4.5: IP adresa

Struktura je zobrazena na obrázku 4.5. V nastavení rozhraní nebo routovací tabulky se používá adresa s maskou (`IpWitchNetmask`), jinde se používá samotná `IpAddress`. IP adresa je vnitřně implementována pomocí integeru, díky této implementaci je možné jednoduše vypočítat broadcast nebo číslo sítě pomocí bitových operací.

4.5.6 MAC adresa

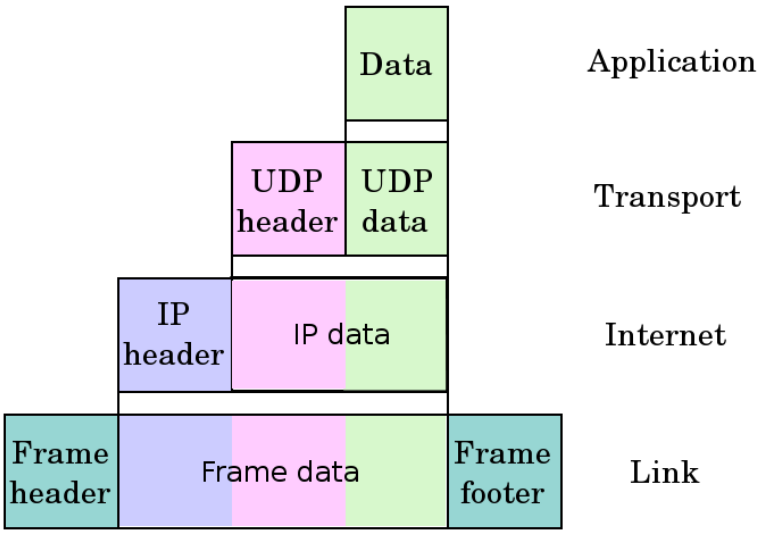
Protokol ethernet používá k adresaci 48-bitovou mac adresu. Ta byla v simulátoru implementována ve třídě `MacAddress`. Protože není potřeba nad ní pro-

vádět žádné výpočty nebo porovnání, je uvnitř reprezentována jako pole bytů, což zjednodušuje její vytváření.

4.6 Pakety v simulátoru

4.6.1 Analýza

Ke komunikaci v internetu se používá mnoho různých protokolů, které jsou dle ISO/OSI modelu rozděleny do vrstev, každá z nich má při přenosu dat svoji funkci. Každý protokol definuje svůj formát rámce. Stejně jako ve skutečnosti i v simulátoru bude použito zapouzdření paketů a to tak, že paket nižší vrstvy bude obsahovat paket vyšší vrstvy jako svoje data, jak je patrno na obrázku 4.6.



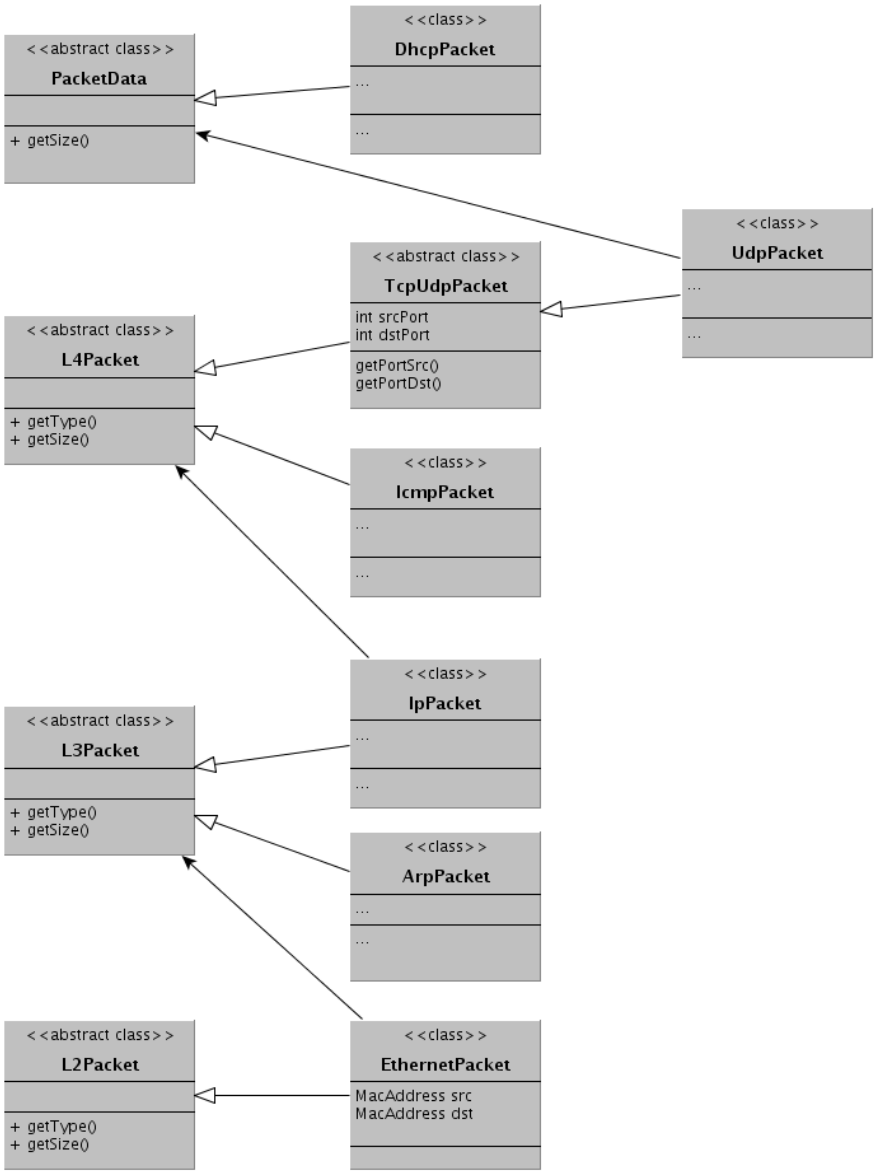
Obrázek 4.6: Zapouzdření paketů, zdroj: [1]

Z analýzy jádra simulátoru (3.3.1) vyplynulo, že se pakety mezi vituálními síťovými zařízeními budou posílat po virtuálních kabelech jako objekty v Javě. Pro účely simulátoru je mnoho parametrů v hlavičkách skutečných paketů zbytečných a jejich vytváření by zabralo zbytečně mnoho času při implementaci. Proto jsme se rozhodli implementovat vlastní zjednodušené pakety. Výhodou tohoto řešení je podstatně jednodušší práce s pakety, nevýhodou je nutnost pakety překládat, pokud jsou posílány z virtuální sítě do reálné.

Pro účely logování byly pakety navrženy tak, aby jednou vytvořený paket již nebylo možné měnit. Jinak by se totiž mohlo stát, že bude zalogován paket, který mezitím nějaké jiné vlákno změní a na standardní výstup nebo ve fron-

tendu by se pak mohly zobrazovat chybné hodnoty. Další výhodou tohoto řešení je to, že při posílání na broadcast je možné posílat jeden paket bez kopírování, bez obavy, že bude jinde změněn.

4.6.2 Realizace paketů v simulátoru



Obrázek 4.7: Struktura všech implementovaných paketů

Pakety byly implementovány dle schématu na obrázku 4.7. `EthernetPacket` jsem implementoval já a podrobně ho popíši v části o linkové vrstvě 4.7.1.2. Pakety protokolu IP, ARP a ICMP implementoval kolega Řehák, těmi se v této práci nezbývám.

4.7 Linková vrstva

Linková vrstva je 2. vrstvou ISO/OSI modelu. Zajišťuje přenos rámců mezi dvěma sousedními systémy. Poskytuje *„funkci spolehlivého spojení (detekci, příp. korekci chyb), formátování dat do rámců, řízení přístupu ke sdílené lince (medium access control), řízení toku na lince a jednoznačnou adresu v rámci segmentu sítě.“*[22]. Z protokolů linkové vrstvy jsem se rozhodl v simulátoru implementovat protokol ethernet, protože je v pevných sítích zdaleka nejrozšířenější a samozřejmě se používá i při laboratorních cvičeních předmětu BI-PSI. Pro potřeby simulátoru, kde nehrozí žádné chyby přenosu a není třeba řídit přístup ke sdílenému médiu, jsem realizoval pouze části protokolu, které zajišťují adresaci, formát rámců a jejich směrování mezi počítači ve stejné síti.

4.7.1 Popis a analýza protokolu ethernet

„Ethernet je souhrnný název pro v současné době nejrozšířenější technologie pro budování počítačových sítí typu LAN (tj. domácí nebo firemní sítě).“[15] Ethernet se stal de facto standardem pro svoji jednoduchost a nízkou cenu a vytlačil z trhu ostatní alternativní technologie (např. ARCNET, ATM, FDDI). Ethernet tedy není jen síťovým protokolem, ale celou množinou technologií. Zasahuje do fyzické a linkové vrstvy ISO/OSI modelu [16]. Je spíše sadou technologií než síťovým protokolem, proto nemá vlastní RFC, byl však standardizován skupinou IEEE 802.3. Dále se pro účely simulátoru zabývám ethernetem pouze jako protokolem linkové vrstvy.

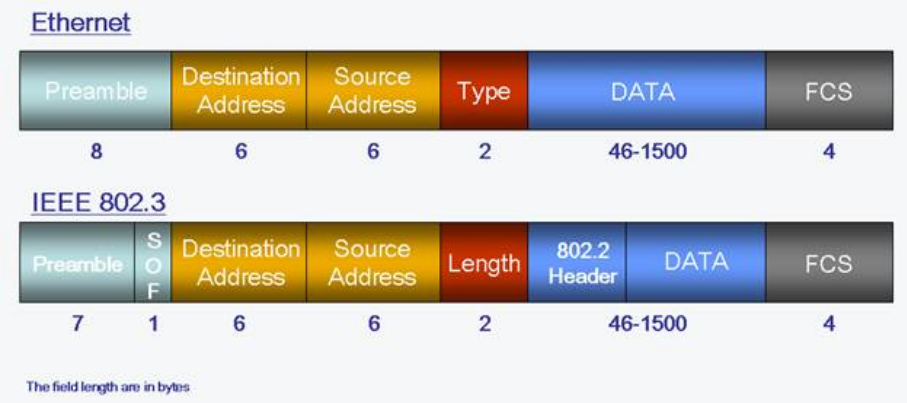
4.7.1.1 Adresace v ethernetu

Protokol ethernet definuje 48-bitovou MAC adresu[11]. Implementace této adresy v simulátoru je popsána v části 4.5.6.

4.7.1.2 Rámce ethernetu

Protože existuje několik různých druhů ethernetu, existuje i několik druhů ethernetových rámců. Já jsem implementoval dnes upřednostňovaný formát rámce ethernet II, který je zobrazen na obrázku 4.8 spolu s formátem IEEE 802.3, od kterého se liší pouze položkou `type`, resp. `length`.

[11] také fyzickou adresu



Obrázek 4.8: Formáty ethernetových rámců

4.7.1.3 Switch

Ethernet předpokládá i sdílení přenosového média, od toho se však v poslední době kvůli rychlosti upouští. Místo toho jsou prvky propojeny pomocí switche, což je aktivní síťový prvek, který dle fyzických adres směruje ethernetové rámce. Jedná se tedy o router na 2. vrstvě ISO/OSI modelu. Switch není třeba konfigurovat, adresování se switch učí automaticky z rámců, které do switche přicházejí, konkrétně z adres odesílatelů. Z těchto údajů si switch automaticky plní tabulku identifikující cílové porty pro jednotlivé adresy. Pokud má switch odeslat rámec na jemu dosud neznámou adresu, chová se jako hub a rozešle rámec do všech ostatních rozhraní. Pravděpodobně na tento rámec časem přijde odpověď, kterou si switch bude moci uložit do tabulky. Pokud by v jedné síti byla smyčka mezi switchi, vedlo by to k nekontrolovanému rozesílání a množení paketů. Tento problém v ethernetu řeší Spanning Tree Protocol, který v síti vypne některé linky tak, aby síť tvořila strom.

4.7.1.4 Ethernetová vrstva switche a routeru

Router i switch provádí směrování, každý ovšem na jiné vrstvě. Z toho plyne i rozdíl mezi síťovým rozhraním switche a routeru, které lze nejlépe popsat na příkladu tzv. „domácího routeru“. „Domácí router“je zařízení, které má například pět portů, z nichž jeden, tzv. WAN port, je připojen do internetu, zatímco do ostatních, tzv. LAN portů, se připojují počítače z místní sítě. Mezi LAN porty domácí router „switchuje“ a mezi všemi LAN porty a WAN portem směruje na síťové vrstvě. Router se pak tváří jako by měl pouze dvě rozhraní, jedno tvoří port WAN a druhé tvoří všechny LAN porty dohromady. Protože jsem chtěl implementovat linkovou vrstvu co nejvíce univerzálně, rozhodl jsem se oddělit klasické síťové rozhraní routeru od síťového rozhraní switche, kterému budu nadále říkat switchport. Mezi switchporty se směruje na linkové

vrstvě, mezi rozhraními na síťové vrstvě. To zpřehlednilo návrh a v budoucnu umožní přidat do simulátoru kromě klasických switchů a routerů i tzv. „domácí routery“.

4.7.1.5 Požadavky na implementaci v simulátoru

Po předchozí analýze jsem se rozhodl v simulátoru v nutné míře implementovat rámce Ethernet II a funkce switche včetně tabulky s adresami. Dle dohody s vedoucím není třeba implementovat STP[12].

4.7.2 Implementace linkové vrstvy

4.7.2.1 Implementace ethernetových rámců

Rozhodl jsem se implementovat rámce Ethernet II (na obrázku 4.8). Implementoval jsem je ve třídě `EthernetPacket`, který dědí od `L2Packetu`, společného předka všech rámců linkové vrstvy. Rámec v simulátoru má zdrojovou a cílovou mac adresu, svoji velikost a odkaz na paket vyšší vrstvy - `L3Packet`.

4.7.2.2 Implementace ethernetové vrstvy síťového modulu

Struktura ethernetové vrstvy síťového modulu je zobrazena na obrázku 4.9. Třída `EthernetLayer` je hlavní třídou této vrstvy, zajišťuje veškerou její činnost. Drží si seznam ethernetových rozhraní `EthernetInterface`, což je třída reprezentující jedno síťové rozhraní. To se dále skládá ze switchportů, mezi kterými se směruje na linkové vrstvě, jak bylo popsáno v 4.7.1.4. K tomu má každé rozhraní tzv. switchovací tabulku, ve které je uloženo, jaké adresy se mají směrovat na jaký switchport. Na začátku je tato tabulka prázdná, postupně se plní jak bylo popsáno v odstavci 4.7.1.3. Třída `SwitchportSettings` je reprezentací switchportu na linkové vrstvě.

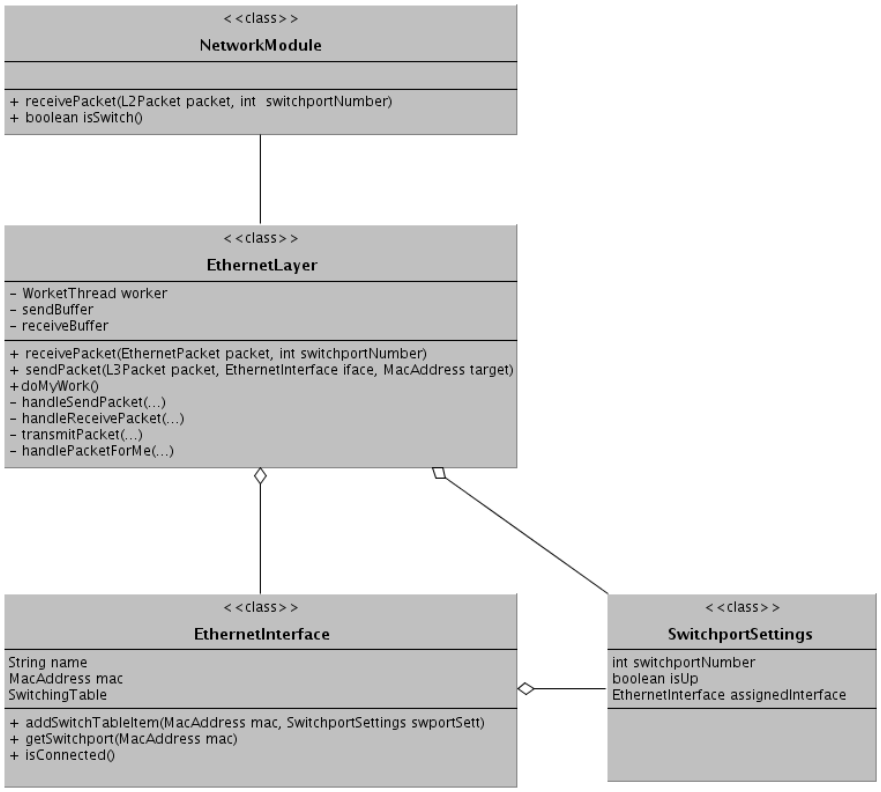
Switch i router mají stejný typ linkové vrstvy. Router má však u každého rozhraní jenom jeden switchport, takže směruje na síťové vrstvě, zatímco switch má pro všechny switchporty jedno rozhraní, a proto směruje na linkové vrstvě.

Vrstva běží ve vlastním vlákně, přijaté pakety jí jsou předávány od fyzického modulu pomocí metody `receivePacket`, pakety k odeslání od síťové vrstvy jsou jí předávány metodou `sendPacket`. Obě tyto metody ukládají pakety do speciálních bufferů, z nich je vyzvedává vlákno ethernetové vrstvy a vyřizuje je.

4.7.3 Možná vylepšení linkové vrstvy

V simulátoru nebyla požadována implementace Spanning Tree Algoritmu, proto, je-li v síti smyčka switchů, pakety se v ní posílají donekonečna. Do

[12]Spanning Tree Protocol



Obrázek 4.9: Struktura ethernetové vrstvy

budoucna by však bylo dobré tento algoritmus implementovat.

4.8 Návrh aplikací a aplikace ping

O aplikacích v simulátoru se zmiňuji jen stručně. Aplikace abstraktně implementovaná třídou `Application` reprezentuje démona běžícího na pozadí, který poslouchá na nějakém portu, přijímá a vyřizuje pakety. Tato podoba aplikace se v budoucnu může hodit pro implementaci například DHCP nebo DNS serveru. Aplikace stejně jako například fyzický modul nebo linková vrstva běží ve vlastním uspávacím vlákně (`WorkerThread`), pakety jí jsou předávány pomocí bufferů.

Pro aplikaci běžící na popředí a aktivně vykonávající nějakou činnost jsme vytvořili třídu `TwoThreadApplication`. Ta dědí od `Application` a kromě toho, že přijímá pakety, vykonává aktivně vlastní činnost v druhém vlákně. Tuto třídu jsme užili v implementaci příkazů `ping` a `traceroute`, které běží déle než okamžik a u kterých je potřeba předat řízení zpět telnet serveru, aby

mohl zachytávat signály jako `ctrl+C`. Tak vznikly třídy `PingApplication` a `TracerouteApplication`.

Implementací aplikací se ve své diplomové práci více zabývá kolega Stanislav Řehák [23].

4.9 Příkazy Linuxu

Příkazy Linuxu byly z větší části převzaty ze starého simulátoru a upraveny tak, aby pracovaly s novým síťovým modulem. Jejich implementací se více zabývám ve své bakalářské práci [18]. Implementovány jsou stejným způsobem, pro každý příkaz je vytvořen objekt a zavolána jeho metoda `run()`, která příkaz zparsuje a provede, případně vypíše informace nebo chybová hlášení. Nově byly přidány příkazy pro manipulaci s filesystémem (`cd`, `ls`, `mv`, `cp`, `cat`, `mkdir`, `mv`, `touch`, `touch` a `rm`). Jejich parser jsem implementoval já a jejich vykonávání kolega Martin Lukáš.

O komunikaci s klientem se stará telnet server, který implementoval taktéž Martin Lukáš. Parser příkazů a doplňování příkazů implementoval abstraktně pro Linux i Cisco kolega Řehák[13] Já implementoval linuxový parser. Návrh je však dílem všech třech.

4.9.1 Linuxový parser příkazů

Parser příkazů je implementovaný ve třídě `LinuxCommandParser`, která dědí od `AbstractCommandParseru`. Parser rozeznává a spouští linuxové příkazy, které je třeba zaregistrovat v jeho metodě `registerCommands`. Parser umožňuje spouštět jednoduché skripty ze souborů uložených ve filesystému. Tyto skripty však mohou obsahovat pouze jednoduché implementované příkazy. Implementace prostředí shellu s proměnnými, řídícími příkazy apod. by převyšovala rozsah diplomové práce.

4.10 Napojení na reálnou síť

Jedním z funkčních požadavků aplikace je možnost posílat ICMP request a reply ze simulátoru do reálné sítě a naopak. Tímto celkem rozsáhlým problémem se zabývám v následující části.

4.10.1 Analýza způsobu napojení

Simulátor poběží na hostitelském počítači. Je tedy třeba propojit nějaký z virtuálních počítačů simulátoru s hostitelským počítačem a to pokud možno tak, aby komunikace mezi těmito dvěma počítači nezpůsobovala nežádoucí

[13] O tom více v jeho diplomové práci [23]

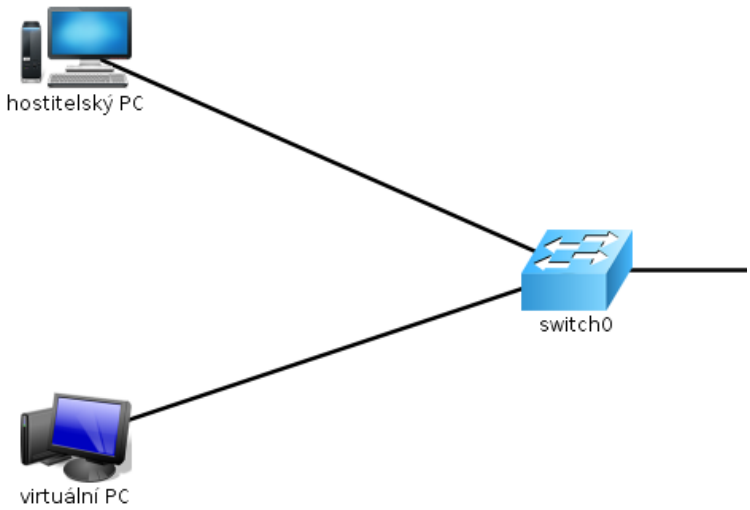
chování jiných síťových prvků ať už reálné sítě, nebo virtuální sítě simulátoru. Reálný a virtuální počítač je nutné propojit minimálně na úrovni linkové vrstvy, proto není možné použít standardní systémové API, které umožňuje pracovat s transportní a síťovou vrstvou [17]. Vedoucím práce mi bylo doporučeno, abych k tomuto napojení použil knihovnu libpcap nebo WinPcap. Obě knihovny, libpcap na linuxu a winpcap na Windows, poskytují aplikacím přístup k síti na linkové vrstvě. Umožňují zachytávat a dekódovat, ale i odesílat kompletní pakety linkové vrstvy přímo na síťových rozhraních [17]. Tyto knihovny používá například známý program Wireshark.

Existuje několik možností, na kterém síťovém rozhraní bude simulátor pomocí některé ze zmíněných knihoven pakety zachytávat a odesílat. Jednou z nich je zachytávání a odesílání paketů na rozhraní, přes které je počítač připojený do internetu. Simulátor by zachytával všechny pakety přicházející do hostitelského počítače a odesílal pakety ze simulátoru. Tato možnost se však ukázala jako velmi nevyhovující z několika důvodů. Simulátor by v tomto případě zachytával veškerou síťovou komunikaci hostitelského počítače, což by ho velmi zatěžovalo. Druhou velkou nevýhodou je fakt, že při tomto řešení by nevzniklo logické propojení mezi hostitelským počítačem a virtuálním počítačem simulátoru, ale pouze mezi virtuálním počítačem a sousedním síťovým zařízením hostitelského počítače, jak je to zobrazeno na obrázku 4.10. Z toho vyplývá, že spojení mezi virtuálním a hostitelským počítačem nebude možné navázat, pokud hostitelský počítač není připojen k internetu.

Protože zachytávání paketů na reálném rozhraní jsem v minulém odstavci vyloučil a více síťových rozhraní na hostitelském počítači nelze předpokládat, jedinou zbývající možností je zachytávání paketů na virtuálním síťovém rozhraní. Pro Linux i pro Windows existují programy, které umí vytvořit virtuální síťovou kartu, se kterou pak jádro operačního systému pracuje stejně jako s klasickou hardwarovou kartou. Virtuální síťová zařízení pracující na linkové vrstvě bývají nazývána TAP rozhraními [20], pro spojení virtuálního počítače s internetem je využívají i jiné programy, jako například QEMU nebo VirtualBox [20]. Používání virtuálního síťového zařízení umožní vytvořit linku přímo mezi hostitelským a virtuálním počítačem, jak je vidět na obrázku 4.11. Na virtuální rozhraní budou chodit pouze pakety určené virtuálnímu počítači, simulátor tak nebude zbytečně zatěžován a komunikace mezi hostitelským a virtuálním počítačem nebude nijak ovlivňovat ostatní prvky sítě.

4.10.2 Wrapper pcap knihoven pro javu

Knihovny libpcap i WinPcap jsou naprogramované v jazyce C a a jejich využití je možné jen v tomto jazyce. Našli jsme však několik wrapperů, které umožňují používat tyto knihovny v java aplikacích. Některý z nich bude muset k práci s pcap knihovnou využívat i náš simulátor.



Obrázek 4.10: Logická struktura sítě při zachytávání paketů na reálném síťovém rozhraní

4.10.2.1 Jpcap z University of California

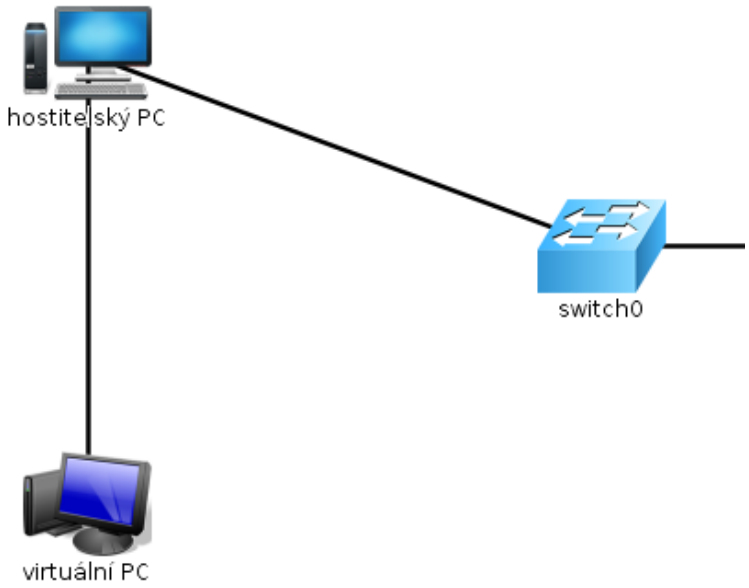
Jpcap je java wrapper pro WinPcap i libpcap [6]. Není moc rozsáhlý a je velmi jednoduchý na použití. Umožňuje nejen dekódovat zachycené pakety, ale i jednoduše pakety tvořit a na síť posílat. Dle webových stránek byl projekt naposledy aktualizován 9.6.2007, odkazovaná diskuze je plná spamu. Z tohoto důvodu jsem se rozhodl tento wrapper nepoužít, protože se jedná o mrtvý projekt a je téměř jisté, že jeho případné bugy již nejsou opravovány.

4.10.2.2 Jpcap ze sourceforge.net

Jedná se o další projekt se stejným jménem. Jeho poslední verze byla vydána roku 2004 [2]. Rozhodl jsem se ho nepoužít ze stejného důvodu jako Jpcap z z University of California.

4.10.2.3 jNetPcap

Tento java wrapper pro libpcap i WinPcap je zajisté nejrozsáhlejší a nejkomplexnější. Je stále vyvíjen a udržován společností Sly Technologies a existuje k němu množství tutoriálů a návodů. Umožňuje jednoduše zachytávat a dekódovat přijaté pakety. Odesílání paketů pomocí tohoto wrapperu je poněkud



Obrázek 4.11: Logická struktura sítě při zachytávání paketů na reálném síťovém rozhraní

složitější, wrapper umožňuje jen úpravu již existujícího paketu nebo jeho vytvoření z pole bytů, což je náročné na implementaci. Přesto jsem se kvůli komplexnosti a poskytované podpoře rozhodl právě pro tuto knihovnu.

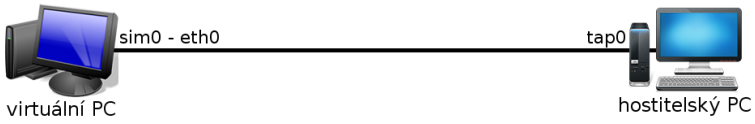
4.10.3 Zachytávání paketů na virtuálním rozhraní ve Windows

Knihovna WinPcap bohužel neumožňuje zachytávat pakety na virtuálních rozhraních. V často kladených otázkách na webových stránkách projektu wireshark stojí:
„Otázka: Podporuje Winpcap loopback device?
Odpověď: Ne, podporována jsou pouze fyzická zařízení. Toto je omezení Windows, nikoliv WinPcap.“ [21]

Z tohoto důvodu jsme se po dohodě s cvičícím rozhodli podporovat připojení do reálné sítě pouze na Linuxu.

4.10.4 Zachytávání paketů na virtuálním rozhraní na Linuxu

Knihovna libpcap umožňuje na rozhraní pakety zachytávat nebo odesílat do sítě. Neumožňuje však posílat pakety jádru OS, tedy „podstrčit“ jádru OS pakety, které ve skutečnosti po kabelu nepřišly. Z toho vyplývá, že jedno virtuální rozhraní k propojení virtuálního a hostitelského počítače nestačí, protože virtuální počítač by neměl možnost posílat pakety hostitelskému počítači. Z tohoto důvodu je nutné na hostitelském počítači vytvořit dvě virtuální rozhraní propojené virtuálním kabelem. Jedno z nich je obsluhováno hostitelským počítačem jako jakékoliv jiné rozhraní. Druhé, zde nazývané `sim0`, nechá hostitelský počítač nenakonfigurované. Propojení virtuální a reálné sítě se naváže tak, že nějaké rozhraní virtuálního počítače (zde nazývané `eth0`) v simulátoru začne zachytávat pakety na rozhraní `sim0`, čímž vlastně dojde k logickému ztotožnění `sim0` a `eth0`, jak je vidět na obrázku 4.12.



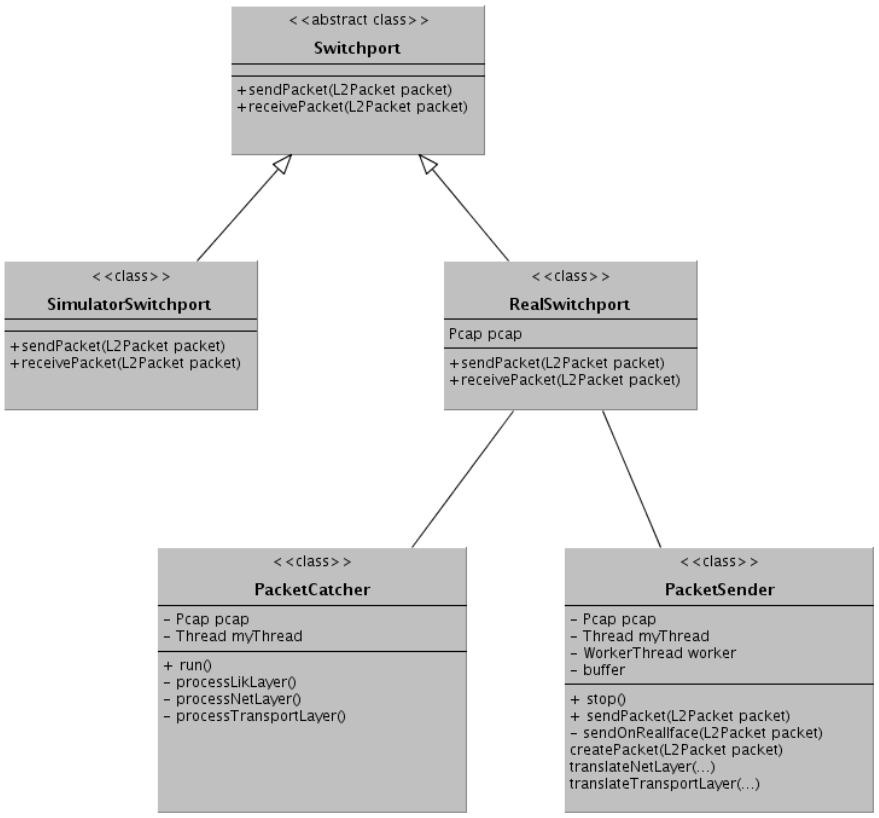
Obrázek 4.12: Dvě virtuální rozhraní pro napojení na reálnou síť. Rozhraní tap0 funguje zcela normálně jako jakékoliv jiné rozhraní, na rozhraní sim0 jsou pakety zachytávány rozhraním eth0 virtuálního počítače.

K vytvoření dvou virtuálních rozhraní a jejich propojení kabelem využívá simulátor programu VDE[14]. K jednoduššímu vytvoření virtuálních rozhraní tímto programem jsem vytvořil skript `virt_iface.sh`, který je dodáván zároveň se simulátorem. K provedení tohoto skriptu, stejně jako k zachytávání paketů, je na Linuxu potřeba administrátorské oprávnění.

4.10.5 Implementace propojení reálné a virtuální sítě

Propojení reálné a virtuální sítě bylo implementováno ve fyzickém modulu dle struktury na obrázku 4.13. K zachytávání paketů má `RealSwitchport` objekt třídy `PacketCatcher`, který zachytává pakety, překládá je na objekty paketů simulátoru a předává dál fyzickému modulu. `PacketCatcher` běží ve vlastním vlákně. K posílání paketů slouží třída `PacketSender`, která přijímá od fyzického modulu `L2Packety`, překládá je na skutečné pakety a odesílá je. Běží ve vlastním uspávacím vlákně (`WorkerThread`). Výhodou této implementace je to, že síťový i fyzický modul pracují s reálným switchportem stejně jako s virtuálním.

[14]Více o VDE na http://vde.sourceforge.net/



Obrázek 4.13: Implementace napojení na reálnou síť

4.10.6 Navázání spojení virtuální a reálné sítě

Propojení reálné a virtuální sítě se nenastavuje ihned při startu serveru dle konfigurace uložené v konfiguračním souboru. Ten totiž musí být přenositelný a na všech počítačích stejně spustitelný. Kdyby při startu simulátoru ještě nebyla na hostitelském počítači vytvořena potřebná virtuální rozhraní, napojení by nemohlo být navázáno a start serveru by se nezdařil. Proto je spojení navazováno až za běhu simulátoru a to speciálním příkazem `rnetconn`[15], který je implementován pouze v simulátoru.

Pro návázání spojení musí tedy uživatel spustit na hostitelském počítači nejprve skript na vytvoření dvojice virtuálních rozhraní:

```
virt_iface.sh pair sim0 tap0
```

Potom musí na libovolném virtuálním počítači simulátoru spustit tento

[15] zkratka slov Real NETwork CONNection

příkaz, který sváže virtuální rozhraní hostitelského počítače `sim0` se switch-portem č. 1 hostitelského počítače `rouer2`:

```
rnetconn tie router2 1 sim0
```

KAPITOLA 5

Testování

5.1 Jednotkové testy

Některé základní datové struktury simulátoru byly otestované jednotkovými testy pomocí JUnit testů. JUnit testy byly vytvořeny pro tyto třídy:

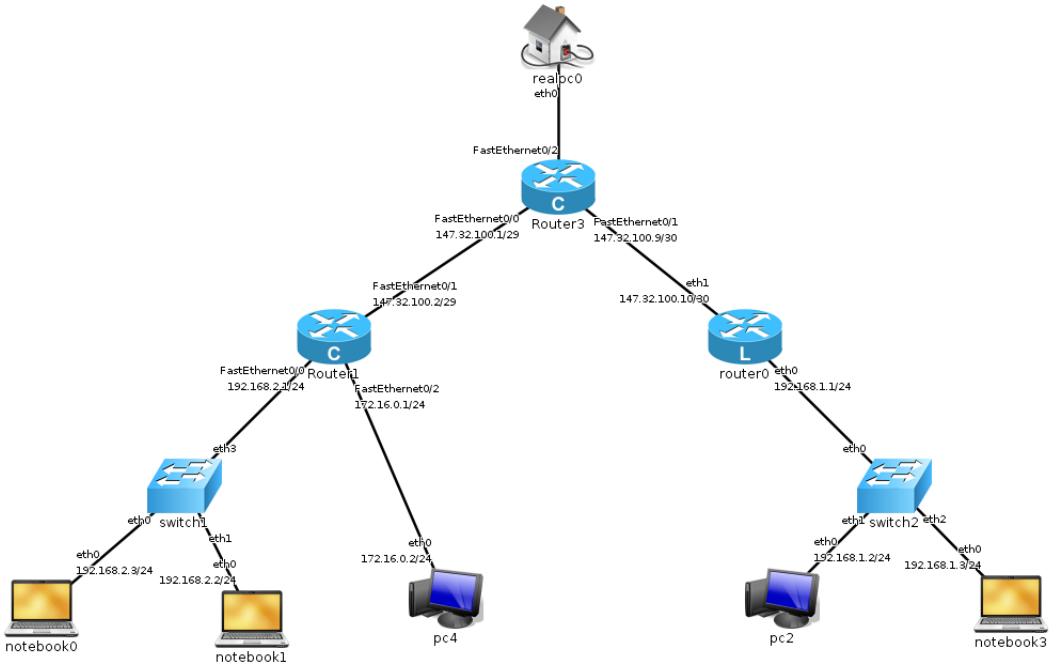
- IP adresa (`IpAddress`)
- IP adresa s maskou (`IpWithNetmask`)
- routovací tabulka (`RoutingTable`)
- MAC adresa (`MacAddress`)

5.2 Uživatelské testování

Uživatelské testování bylo prováděno studenty na cvičeních BI-PSI během letního semestru 2012, především při cvičeních 11. dubna. Testován byl frontend i backend, dále se budu zabývat pouze testováním backendu.

5.2.1 Průběh uživatelského testování backendu

Studenti dostali konfigurační soubor zatím nenakonfigurované virtuální sítě, kterou měli za úkol dle zadaných pokynů nakonfigurovat. Síť je zobrazena na obrázku 5.2. Celá úloha byla záměrně vytvořena tak, aby se co nejvíce podobala úlohám, které studenti BI-PSI řešili minulá cvičení na skutečném hardwaru. Studenti dostali formuláře, na které měli psát odhalené chyby a navrhovaná zlepšení, případně další připomínky.



Obrázek 5.1: Síť použitá při uživatelském testování

5.2.1.1 Zadané úkoly při konfiguraci testovací sítě

Ke konfiguraci testovací sítě dostali studenti tyto pokyny:

1. nastavte IP adresy na pc2, router0
2. zprovozněte spojení mezi pc2 a router0 (pozor na správné routy)
3. nastavte IP na router3 pro FA0/1 a nastavte maškarádu na router0 pro pc2
4. nakonfigurujte IP adresy na notebook1, pc4, router1
5. nakonfigurujte statický překlad adres pro prvek pc4 na router1
6. nakonfigurujte dynamický překlad adres pro prvek notebook1 na router1
7. správné nastavení obou překladů adres ověřte pingem z notebook1 a pc4 na router3 (pozor na nastavení routovacích tabulek)

5.2.2 Nalezené chyby a připomínky k backendu a jejich vyhodnocení

1. Windows telnet nereagoval na ENTER

Problém je způsoben rozdílnými bajty pro konec řádku na Linuxu a na Windows. Jde o práci Martina Lukáše a nechám na něm, jestli tuto připomínku zapracuje.

2. nefunguje příkaz `save` na Ciscu

   Příkaz již funguje.

3. nefunguje doplňovaní na Ciscu "interface FA0/0"

   Týká se práce kolegy Řeháka.

4. zkontrolovat linuxovou maškarádu

   Maškaráda byla zkontrolována a funguje správně.

5. nefunguje příkaz: `route -en`

   Příkaz byl opraven.

6. Několika lidem vůbec nešlo spustit simulátor a vypisovalo se hlášení `psimulator2.Main was not found`.

   Problém je způsoben tím, že uživatel spustil simulátor pod JRE verze 1.6. Program je však spustitelný pouze pod JRE 1.7, což je v instalační příručce uvedeno. Uživatel ji tedy zřejmě nečetl. V brzké době však bude nová verze JRE distribuována v rámci aktualizací a tento problém tedy brzo vymizí.

7. Frontend by mohl umožňovat přidávání výchozí brány.

   Simulátor byl vyvinut mimo jiné proto, aby se studenti naučili konfigurovat počítačovou síť v příkazové řádce. Tento požadavek je tudíž dle mého názoru nesmyslný. IP adresy jsou v simulátoru zobrazovány pouze pro větší přehlednost.

8. Zjednodušit spuštění backendu.

   Možnost spustit backend přímo z frontendu by byla velmi užitečná a bylo by dobré ji časem doimplementovat. Je to však záležitost frontendu a tedy kolegy Švihlíka.

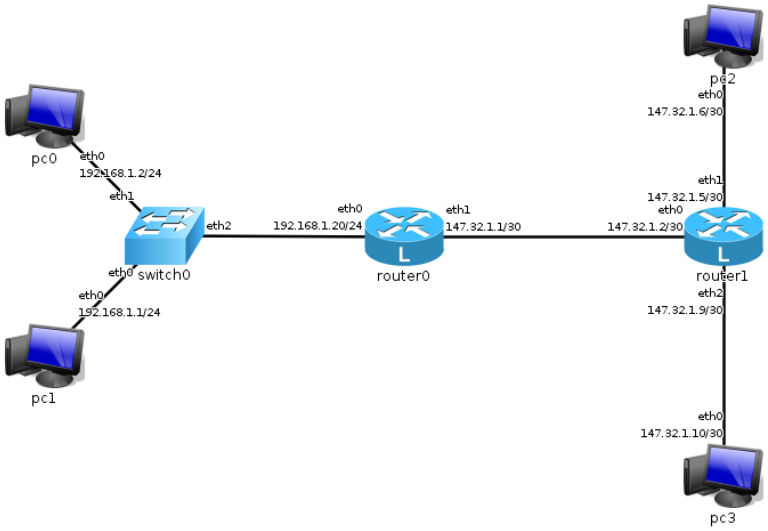
9. Na Linuxu nefungoval ping na loopback (na adresu 127.0.0.1).

   Tato funkcionalita byla doplněna.

5.3 Automatické testování

Systém pro automatické testování simulátoru implementoval kolega Martin Lukáš a popisuje ho ve své diplomové práci [8]. Já s kolegou Řehákem jsme k tomu dodali testovací síť, která je zobrazena na obrázku 5.2, a scénáře testování.

Počítače byly nakonfigurovány tímto způsoben:



Obrázek 5.2: Síť použitá při uživatelském testování

- **pc1**: výchozí brána na router0
- **pc0**: nemá defaultní bránu
- **router0**: maškaráda na odchozí rozhraní eth1, výchozí brána na router1
- **router1**: žádná maškaráda ani výchozí brána, brána na 89.190.94.0/24 na pc2, brána na 147.32.125.128/25 na 147.32.1.10
- **pc2**: chybně nastavený počítač, má výchozí bránu na router1 (pro testování `time to live exceeded`)
- **pc3**: chybně nastavený počítač, nemá žádnou defaultní bránu (pro testování, kdy se nemá vrátit nic)

5.3.1 Scénář k automatickému testování počítačů s Linuxem

Na počítači `pc1` vykonat tyto příkazy:

- `ping -c1 192.168.1.2` (očekává `ICMP reply`)
- `ping -c1 192.168.1.20` (očekává `ICMP reply`)
- `ping -c1 192.168.1.3` (očekává `destination host unreachable` od 192.168.1.1)

- `ping -c1 8.8.8.8` (očekává `destination net unreachable` od 147.32.1.2)
- `ping -c1 89.190.94.1` (očekává `time to live exceeded`)
- `ping -c1 147.32.125.234` (neočekává nic, žádná odezva)
- `ping -c1 147.32.1.6` (očekává `ICMP reply`)
- `ping -c1 147.32.1.5` (očekává `ICMP reply`)

Na počítači `pc0` vykonat tento příkaz:

- `ping -c1 8.8.8.8` (očekává `connect: Network is unreachable`)

KAPITOLA 6

Další navrhovaná rozšíření a vylepšení

Backend simulátoru by mohl být dále vylepšován a rozšiřován. Zde uvádím několik návrhů:

- Pro jednodušší spuštění backendu by bylo vhodné implementovat možnost jeho spuštění z frontendu.
- Do simulátoru může být přidána nová aplikace. Doporučuji implementovat například DNS server, jednoduchý HTTP server nebo dokončit již z části implementovaný DHCP server.
- Architektura simulátoru umožňuje vytvořit nový síťový prvek nebo vyměnit stávající síťový modul. Mohl by tak být přidán například hub nebo „domácí router“ (více v 4.7.1.4.
- Logování by bylo možné využít k implementaci linuxového příkazu `tcpdump`.
- Na linkové vrstvě by bylo dobré implementovat STP[16].
- Pro větší přehlednost by bylo dobré, aby si uživatel sám mohl nastavit úrovně logování pro různé kategorie logovacích zpráv. Toto vylepšení by ale pravděpodobně ocenila jen malá část uživatelů.

6.1 Návod na přidání aplikace

Do simulátoru může být implementována nová aplikace, která na virtuálním počítači umožní spustit například DNS, HTTP nebo DHCP server nebo klient.

[16] Spanning Tree Protocol

K vytvoření aplikace, která komunikuje po síti a reaguje na příchozí požadavky (např. DHCP server), je třeba vytvořit novou třídu, která dědí od třídy `Application`, a v ní implementovat všechny abstraktní metody třídy `Application`. Metody jsou zdokumentované přímo v kódu pomocí javadocu. Je pouze nutné si uvědomit, že aplikace běží ve vlastním uspávacím vlákně, a proto veškeré požadavky je třeba jí předávat pomocí bufferů a probouzet její vlákno.

Pro přidání aplikace, která reaguje na požadavky ze sítě a navíc sama vykonává nějakou činnost, je třeba vytvořit třídu, která dědí od třídy `TwoThreadApplication`, a v ní implementovat všechny abstraktní metody této třídy. Příkladem, podle kterého lze novou aplikaci implementovat, je aplikace `Ping`.

Aplikace jsou podrobněji popsány především v diplomové práci Stanislava Řeháka [23] a v části 4.8 této diplomové práce.

Kromě samotného přidání aplikace je také nutné vytvořit příkaz k jejímu spuštění a zaregistrovat ho v parseru (o tom v 4.9). Pokud aplikace používá síťový protokol, který zatím není v simulátoru implementován (např. DNS, HTTP), je samozřejmě nutné implementovat také tento protokol. Podrobnější informace k tomuto problému jsou v 4.6 a v diplomové práci kolegy Řeháka [23].

Závěr

V rámci diplomového projektu jsme společně s kolegy Stanislavem Řehákem, Martinem Lukášem a Martinem Švihlíkem navrhli a implementovali multiplatformní síťový simulátor, který dokáže simulovat virtuální počítačovou síť složenou z počítačů s OS Linux, směrovačů s Cisco IOS a jednoduchých switchů. Simulátor má intuitivní grafické uživatelské rozhraní, je jednoduchý na použití a umožňuje konfiguraci síťových zařízení v režimu příkazové řádky pomocí protokolu telnet. Síťovou komunikaci virtuální sítě je možné vizualizovat v grafickém rozhraní. Virtuální síť simulátoru je možné propojit s reálnou sítí tak, že je možné posílat ping z reálné sítě do virtuální a naopak. Simulátor byl navržen tak, aby bylo v budoucnu možné přidávat další síťová zařízení, síťové protokoly nebo aplikace. Byly splněny všechny požadavky ze zadání této diplomové práce i požadavky, které vyplynuly ze zkušeností s používáním staré verze simulátoru.

Celý projekt obsahuje 60 000 řádků kódu v Javě, samotný backend má asi 35 000 řádek.

Po vystavení projektu na serveru Google Code nás kontaktoval učitel španělské university La Laguna [17] se zájmem využít simulátor při výuce počítačových sítí. V současné době je simulátor na této univerzitě testován.

Již stará verze simulátoru se osvědčila jako pomůcka při výuce předmětu PSI. Proto si myslím, že i nový simulátor s mnoha novými funkcemi a grafickým uživatelským rozhraním bude pro tento předmět přínosem.

[17] Universidad de La Laguna `http://www.ull.es/`

Literatura

[1] Boerner, G.: *Internet Protocol*. [cit. 2012-05-01]. Dostupné z WWW: <http://www.boerner.net/jboerner/?cat=22&paged=12>

[2] Charles, P.: *Jpcap*. [cit. 2012-05-02]. Dostupné z WWW: <http://sourceforge.net/projects/jpcap/>

[3] Cisco systems: *Cisco Packet Tracer*. [cit. 2012-04-25]. Dostupné z WWW: <http://www.cisco.com/web/learning/netacad/course_catalog/PacketTracer.html>

[4] GNS3 Community: *Dynamips*. [cit. 2012-04-25]. Dostupné z WWW: <http://www.gns3.net/dynamips/>

[5] GNS3 Community: *GNS3*. [cit. 2012-04-25]. Dostupné z WWW: <http://www.gns3.net/>

[6] Jpcap developers group: *Jpcap*. [cit. 2012-05-02]. Dostupné z WWW: <http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/index.html>

[7] Kotala, Z.: *Vlákna*. 2012. Dostupné z WWW: <http://v1.dione.zcu.cz/java/sbornik/16.html>

[8] Lukáš, M.: *Podpůrné komponenty simulátoru počítačové sítě: Diplomová práce*. Praha: ČVUT v Praze, Fakulta informačních technologií, 2012.

[9] Mašek, J.: *Simulační úlohy v NS2 osvětlující znalosti z předmetu MPKT: Bakalářská práce*. Brno: Vysoké učení technické v Brně, 2010.

[10] Michek, J.: *Emulátor počítačové sítě: Diplomová práce*. Praha: ČVUT v Praze, Fakulta informačních technologií, 2008.

[11] ns-2: *The Network Simulator - ns-2*. [cit. 2012-04-25]. Dostupné z WWW: <http://www.isi.edu/nsnam/ns/>

[12] Odvárka, P.: *ICMP*. [cit. 2012-05-01]. Dostupné z WWW: <http://www.svetsiti.cz/clanek.asp?cid=ICMP-Internet-Control-Message-Protocol-1012001>

[13] Omnet++ Community: *Omnet++*. [cit. 2012-04-25]. Dostupné z WWW: <http://www.omnetpp.org/>

[14] Oujezský, V.: *Možnosti Cisco emulátoru sítí GNS3*. [cit. 2012-04-25]. Dostupné z WWW: <http://computerworld.cz/testy/test-moznosti-cisco-emulatoru-siti-gns3-5632>

[15] Pařízek, A.: *Softwarové nástroje pro směrování v sítích TCP/IP: Bakalářská práce*. Praha: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, 2011.

[16] Peterka, J.: *Ethernet*. [cit. 2012-05-01]. Dostupné z WWW: <http://www.earchiv.cz/i_prednasky.php3>

[17] Pintér, D.: *Knihovna Winpcap a její základní použití*. [cit. 2012-05-02]. Dostupné z WWW: <http://www.root.cz/clanky/knihovna-winpcap-a-jeji-zakladni-pouziti/>

[18] Pitřinec, T.: *Simulátor virtuální počítačové sítě Linux: Bakalářská práce*. Praha: ČVUT v Praze, Fakulta elektrotechnická, 2010.

[19] Přispěvatelé Wikipedie: *ns (simulator)*. [cit. 2012-04-25]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Ns_%28simulator%29>

[20] Přispěvatelé Wikipedie: *TUN/TAP*. [cit. 2012-05-02]. Dostupné z WWW: <http://de.wikipedia.org/wiki/TUN/TAP>

[21] Riverbed Technology: *WinPcap*. [cit. 2012-05-02]. Dostupné z WWW: <http://www.winpcap.org/misc/faq.htm>

[22] Smotlacha, V.: *Počítačové sítě*. [cit. 2012-05-01]. Dostupné z WWW: <https://edux.fit.cvut.cz/courses/BI-PSI/_media/lectures/p1-uvod.pdf>

[23] Řehák, S.: *Síťový simulátor pro výukové účely na bázi směrovačů CISCO: Diplomová práce*. Praha: ČVUT v Praze, Fakulta informačních technologií, 2012.

PŘÍLOHA A

Seznam použitých zkratek

API Application Programming Interface

ARP Address Resolution Protocol

ATM Asynchronous Transfer Mode

BI-PSI předmět Počítačové sítě na FIT ČVUT

Cisco IOS Cisco Internetwork Operating System

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

FDDI Fiber distributed data interface

FTP File Transfer Protocol

GUI Graphical User Interface

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDE Integrated Development Environment

IP Internet Protocol

Java SE Java Standard Edition

JRE Java Runtime Environment

LAN Local Area Network

MAC Media Access Control

NAT Network Address Translation

OS Operační systém

PC Personal Computer

RFC Request for Comments

STP Spanning Tree Protocol

TCP Transmission Control Protocol

TTL Time To Live

UDP User Datagram Protocol

UML Unified modeling language

VDE Virtual Distributed Ethernet

WAN Wide Area Network

XML Extensible markup language

PŘÍLOHA B

Instalační a uživatelská příručka

B.1 Systémové požadavky

B.1.1 Java Runtime Environment version 7+

Pro běh simulátoru je nutné stáhnout JRE verze 7 nebo vyšší:
`http://www.oracle.com/technetwork/java/javase/downloads/`

B.1.2 Telnet klient

Pro připojení k backendu je možné použít zabudovaného telnet klienta nebo je možné stáhnout putty:
`http://www.putty.org/`.

B.1.3 Napojení na skutečnou síť

Propojení virtuální sítě se skutečnou je možné pouze na unixových systémech. Nainstalujte si:

- `libpcap` z repozitáře
- `VDE virtual switch` z repozitáře (v Debianu je to balík `vde2`)

B.2 Instrukce pro instalaci

Stáhněte a rozbalte `psimulator2`:
`http://code.google.com/p/psimulator/downloads/list`

B.3 Instrukce pro spuštění

B.3.1 Vytvoření sítě

Pro spuštění simulátoru je nutné mít k dispozici konfigurační soubor sítě, kterou chceme simulovat. Pro vytvoření sítě spusťte frontend příkazem v konzoli:

```
java -jar psimulator2_frontend.jar
```

B.3.2 Spuštění backendu

Po vytvoření sítě je možné spustit backend příkazem:

```
java -jar psimulator2_frontend.jar konfigurační_soubor
```

Pro vizualizaci paketů je možné propojit backend s frontendem. Ve spuštěném frontendu klikněte na `Připojit k serveru`.

B.3.3 Připojení na síťový prvek

Backend vypíše seznam nastartovaných síťových prvků s čísly portů, na kterých poslouchají. Pro připojení k libovolnému prvku spusťte telnet klienta na localhost s portem prvku, ke kterému se chcete připojit, např.:

```
telnet localhost 11001
```

Pokud je frontend napojen na backend, tak je možné provádět konfiguraci prvků z integrovaného telnet klienta ve frontendu: vpravo nahoře přepněte na záložku Simulátor a po kliknutí pravým tlačítkem na libovolný (konfigurovatelný) prvek je možné otevřít telnet spojení na vybraný prvek.

B.3.4 Napojení na skutečnou síť

K napojení simulátoru na skutečnou síť je na hostitelském počítači nutné superuživatelské oprávnění.

B.3.4.1 Vytvoření virtuálních rozhraní

Nejprve je třeba na hostitelském počítači vytvořit pár propojených virtuálních rozhraní dodaným skriptem `virt_iface.sh`:

```
./virt_iface.sh pair sim0 tap0
```

Je nutné mít nainstalovaný program VDE.

B.3.4.2 Svázání virtuálního rozhraní hositelského PC se switchportem v simulátoru

K navázání komunikace je třeba příkazem `rnetconn` svázat switchport simulátoru s virtuálním rozhraním hostitelského počítače. Příkaz je implementován na všech síťových prvcích simulátoru, na kterých běží telnet. Příkazem je možno spravovat napojení na reálnou síť u všech počítačů, nejen na tom, na

kterém je příkaz spuštěn. Pro vypsání seznamu všech switchportů ve virtuální síti, které jsou určeny k napojení na reálnou síť, spusťte příkaz:
`rnetconn list`

Pro svázání rozhraní `sim0` hostitelského počítače se switchportem č. 1 na virtuálním počítači router2 spusťte příkaz:
`rnetconn tie router2 1 sim0`

K tomu je potřeba mít nainstalovanou knihovnu libpcap.

Zbývá již pouze nakonfigurovat rozhraní na hostitelském i virtuálním počítači.

PŘÍLOHA C

Obsah přiloženého CD

Přiložené CD obsahuje tyto soubory a adresáře:

`readme.txt`	informace o souborech na CD
`text/`	adresář s textem práce
`text/DP_pitritom`	text práce ve formátu PDF
`text/latex/`	zdrojová forma textu ve formátu LATEX
`bin/`	adresář se spustitelnými a konfiguračními soubory
`javadoc/`	dokumentace javadoc
`source/`	zdrojové kódy
`source/config`	konfigurační soubory použité pro testování