České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového inženýrství

Diplomová práce

Síťový simulátor pro výukové účely na bázi směrovačů CISCO

Bc. Stanislav Řehák

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

8. května 2012

Poděkování

Nejprve bych chtěl poděkovat své rodině a Pétě, že vytvořili zázemí, které mi bylo oporou. Dále bych chtěl poděkovat vedoucímu práce Ing. Pavlovi Kubalíkovi, Ph.D. za užitečné rady a připomínky při tvorbě této práce. Mé poděkování patří také všem korektorům, testerům a dalším lidem, kteří mi nějakým způsobem pomohli.

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

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

Abstract

This master thesis inquires into desing and implementation of network simulator composed of cisco-based routers for subject BI-PSI Computer networks. Simulator allows configuration of routers via Cisco IOS command line. Simulator implements commands for configuring interfaces, static routing, dynamic and static network address translation. Simulator system also offers verification of current configuration via ping and traceroute commands. Entire network settings are loaded/saved from/to a configuration file. Simulator implements basic versions of following protocols Ethernet II, IP, ARP, ICMP and UDP. User testing is integral part of this thesis.

Keywords

network simulator, computer networks, network protocols, Cisco IOS, router.

Abstrakt

Tato práce se zabývá návrhem a implementací simulátoru počítačové sítě založené na směrovačích od firmy Cisco Systems pro předmět BI-PSI Počítačové sítě. Simulátor umožňuje konfiguraci cisco směrovače přes textové rozhraní Cisco IOS. Simulátor implementuje příkazy pro konfiguraci rozhraní, statické směrování, dynamický a statický překlad adres. Pro ověření správnosti konfigurace jsou implementovány příkazy ping a traceroute. Celá konfigurace virtuální počítačové sítě se načítá ze souboru a je možné ji následně uložit zpět. Simulátor dále implementuje základní verze protokolů Ethernet II, IP, ARP, ICMP a UDP. Součástí práce je uživatelské testování celého simulátoru.

Klíčová slova

síťový simulátor, počítačové sítě, síťové protokoly, Cisco IOS, router.

Obsah

Seznam obrázků

Úvod

V předmětu Počítačové sítě (BI-PSI) na Fakultě informačních technologií ČVUT se při laboratorních cvičeních konfiguruje síť složená z několika směrovačů založených na linuxu a dalších směrovačích od Cisco Systems. Studenti stráví při laboratorních cvičeních většinu času řešením jednoduchých problémů a na skutečné experimenty v sítích nezbývá mnoho času. Studenti zpravidla nemají s konfigurací sítí zkušenosti a na laboratorní cvičení chodí nepřipraveni.

Během své bakalářské práce na FEL ČVUT jsem vytvořil s kolegou Bc. Tomášem Pitřincem jednoduchý síťový simulátor pro potřeby předmětu Y36PSI (obdoba předmětu Počítačové sítě vyučovaného na Fakultě elektrotechnické). Od té doby je tento simulátor používán při výuce. Simulátor se osvědčil, nicméně postrádá uživatelské rozhraní pro tvorbu sítě a má několik nedostatků jako např. absence robustního ISO/OSI modelu nebo nevhodně řešený ARP protokol, dále simulátor podporuje pouze jeden typ paketů - ICMP.

Tato diplomová práce se zabývá návrhem a implementací nové verze síťového simulátoru pro potřeby předmětu BI-PSI, která projekt psimulator1 vylepší o řadu funkcionalit dále popsaných v kapitole 1.2.

KAPITOLA 1

Popis problému, specifikace cíle

1.1 Problémy aktuální verze simulátoru

Na cvičeních předmětu Počítačové sítě (BI-PSI) na Fakultě informačních technologií při ČVUT se používá simulátor, který jsem vytvořil společně s kolegou Bc. Tomášem Pitřincem v rámci našich bakalářských prací. Na simulátoru je možné testovat konfiguraci rozhraní, statické směrování a překlad adres (dynamický i statický). Nakonfigurovanou síť je možné otestovat příkazem `ping`. Serverová konzole vypisuje informace o průchodu paketů jednotlivými prvky simulátoru. Simulátor vyhovoval původním požadavkům, nicméně po nasazení programu do výuky vyvstaly nové požadavky.

Simulátor obsahuje několik zásadních omezení, která brání rozšíření některých funkcionalit:

- Celý simulátor běží prakticky v jediném vlákně (připojení k serverové části = jedno vlákno). Když je posílán jeden paket, tak tento paket musí projít celou sítí a pak teprve je možné odeslat další. Tato situace není udržitelná, pokud chceme mít reálnější simulace.

- Absence oddělených vrstev ISO/OSI modelu, které by nezávisle na sobě obsluhovaly požadavky (týká se především příchozích a odchozích paketů).

- Simulátoru de facto chybí oddělená linková vrstva, která by pracovala nezávisle na síťové. Tato vrstva je sice v simulátoru emulována, bohužel už ale neumožňuje rozšíření sítě o switche či huby.

- Výpisy ze serverové konzole jsou nepřehledné, a proto je těžké sledovat, k jakým akcím v simulátoru dochází. Tento problém by mohlo řešit grafické uživatelské rozhraní, které by umělo vizualizovat posílané pakety mezi jednotlivými síťovými prvky a zobrazovat obsahy těchto paketů.
- Stavba topologie sítě je velmi problematická, protože je nutné ji napsat ručně v jazyce XML. Tento problém by mohl řešit grafický nástroj, který by umožňoval vytváření a úpravu sítě.
- Simulátor podporuje pouze jediný typ paketů - ICMP. Rozšíření o další typy by bylo velmi složité a vyžadovalo by hluboký zásah do simulátoru.
- Napovídání příkazů je řešeno přes externí program (`rlwrap`). Tento napovídací systém nerozlišuje ani mezi stavy Cisco IOSu ani stavu v rámci jednoho příkazu.
- Simulátor nepodporuje klávesové zkratky Ctrl+C, Ctrl+Z a další, které jsou velmi užitečné při konfiguraci síťových prvků.
- Historie příkazů je rovněž řešena přes `rlwrap`, a proto nebere v úvahu módy (historie příkazů jsou oddělené podle módů Cisco IOS).
- Simulátor podporuje posílání/přijímání pouze celé řádky od klientů (nepodporuje posílání/přijímání po znaku). Řešením tohoto problému by mohlo být vložení knihovny, která by implementovala telnet protokol.

1.2 Specifikace cíle

Cílem práce je navrhnout a implementovat novou verzi simulátoru pro potřeby předmětu BI-PSI. Tento projekt bude rozdělen na grafickou část - frontend a jádro - backend. Má práce se bude zabývat pouze backendem. Podrobnější rozdělení práce viz kapitola 1.2.3.

Nová verze simulátoru pravděpodobně převezme některé části ze stávající verze (psimulator1 [31]) (bude se týkat především parserů příkazů a části datových struktur).

Nový simulátor by měl mít robustní architekturu, aby dokázal vyřešit všechny na něj kladené požadavky.

1.2.1 Funkční požadavky

1. Simulátor musí umožnit vytvoření počítačové sítě pomocí grafického nástroje.
2. Simulátor musí umožnit konfiguraci rozhraní přes příkazovou řádku.
3. Simulátor musí umožnit simulaci směrování paketů.

4. Simulátor musí implementovat dynamický i statický překlad adres.
5. Simulátor musí podporovat ukládání a načítání sítě do/ze souboru.
6. Pro ověření správnosti musí být implementovány příkazy ping a traceroute.
7. K jednotlivým síťovým prvkům (tam kde to má smysl) musí být umožněno připojení pomocí telnetu.
8. Pomocí telnet klientů musí být možné několikanásobné paralelní připojení k jednomu síťovému prvku v jeden okamžik.
9. Simulátor musí podporovat reakce na signály Ctrl+C, Ctrl+Z a apod.
10. Simulátor musí nabízet příkazy z historie (správně dle módů).
11. Simulátor musí doplňovat příkazy přes klávesu TAB.
12. Simulátor musí obsahovat alespoň základní implementaci linkové vrstvy.
13. Simulátor musí umožnit zapojit do sítě tyto prvky: směrovač Cisco (pouze podmnožina příkazů a části systému), jednoduchý switch a směrovač založený na Linuxu
14. Simulátor musí umožňovat spouštění dlouhodobě běžících aplikací (služeb, daemonů).
15. Simulátor musí v potřebné míře implementovat protokoly ARP[1], IPv4, ICMP a UDP.

1.2.2 Nefunkční požadavky

1. Simulátor musí být navržen s ohledem na budoucí rozšiřitelnost o další funkcionality.
2. Simulátor musí být multiplatformní - alespoň operační systémy Windows, Linux a MacOS X
3. Simulátor musí být spustitelný na běžném počítači bez enormních nároků na hardware.
4. Simulátor se musí snažit být co nejvěrnější kopií skutečného cisca v mezích zadání a potřeb předmětu BI-PSI.

[1] ARP - Address Resolution Protocol

1.2.3 Rozdělení práce

Protože se jedná o projekt velkého rozsahu a bude na něm pracovat tým lidí, tak jsme ho s kolegou[2] rozdělili na dvě logické části:

- backend - serverová část simulátoru, která bude obstarávat simulace (přeposílání paketů mezi prvky, zpracování paketů v rámci prvků)
- frontend - grafické uživatelské rozhraní pro vytváření a úpravu topologie sítě + grafická vizualizace průchodu paketů sítí

Tyto dvě části jsme rozdělili mezi tyto čtyři členy týmu:

1.2.3.1 Backend

Backend budu mít na starosti hlavně já s kolegou Bc. Tomášem Pitřincem, protože máme zkušenosti z první verze našeho simulátoru. Do backendu budeme potřebovat moduly, které bude zajišťovat Bc. Martin Lukáš.

Stanislav Řehák (já)

- návrh a implementace jádra simulátoru
- návrh napojení na skutečnou síť
- návrh a implementace částí fyzické vrstvy
- síťová vrstva: ARP protokol
- síťová vrstva: IP protokol (bez routovací tabulky)
- síťová vrstva: dynamický i statický překlad adres (NAT)
- transportní vrstva
- návrh a implementace aplikací
- aplikace ping a traceroute
- Cisco IOS: parser příkazů + příkazy
- simulace bufferů na rozhraních, zpoždění paketů

[2] Bc. Tomáš Pitřinec

Tomáš Pitřinec

- návrh a implementace jádra simulátoru
- návrh a implementace napojení na skutečnou síť
- návrh a implementace částí fyzické vrstvy
- návrh a implementace linkové vrstvy
- síťová vrstva: směrovací tabulka (s úpravami se použije z aktuální verze simulátoru)
- architektura aplikací
- linuxový parser příkazů a linuxové příkazy
- napojení na reálnou síť (pro ICMP)

Martin Lukáš

- integrace telnet protokolu
- souborový systém pro Linux
- ukládání a načítání topologie sítě a konfigurace všech síťových prvků do XML
- napojení backendu simulátoru na grafické simulační rozhraní
- jednoduchý textový editor v terminálovém rozhraní

1.2.3.2 Frontend

Martin Švihlík

- vytváření a úprava topologie sítě
- automatické rozmístění síťových prvků
- vizualizace průchodu paketů sítí
- vypisování paketů přijatých z backendu

Martin Lukáš

- grafický telnet klient

KAPITOLA 2

Existující řešení

Rešerše je založena na poznatcích mé bakalářské práce z roku 2010: Simulátor virtuální počítačové sítě Cisco [34], kde popisuji existující řešení pro potřeby výuky předmětu Počítačové sítě (Y36PSI) vyučovaný na Elektrotechnické fakultě ČVUT.

Existujících simulátorů je celá řada, bohužel je ale většina z nich nevhodná pro výukové účely.

2.1 NS-3

Nejznámějším síťovým simulátorem je projekt NS-3 [27](3. verze NS). Tento simulátor je často používán kvůli širokým možnostem u simulací směrovacích protokolů a ve výzkumu ad-hoc sítí. Simulátor NS-3 nedisponuje žádným grafickým rozhraním. Uživatel si napíše pomocí skriptovacího jazyku topologii sítě, v příkazovém řádku spustí experimenty a výsledná data vyhodnotí externím programem. Nevýhodou NS-3 simulátoru je velmi komplexní a složitá konfigurace sítě, jinými slovy tu citelně chybí jednoduché uživatelské rozhraní. Simulátor je dostupný pro Linux, pro Windows neexistuje nativní verze (pouze verze přes cygwin).

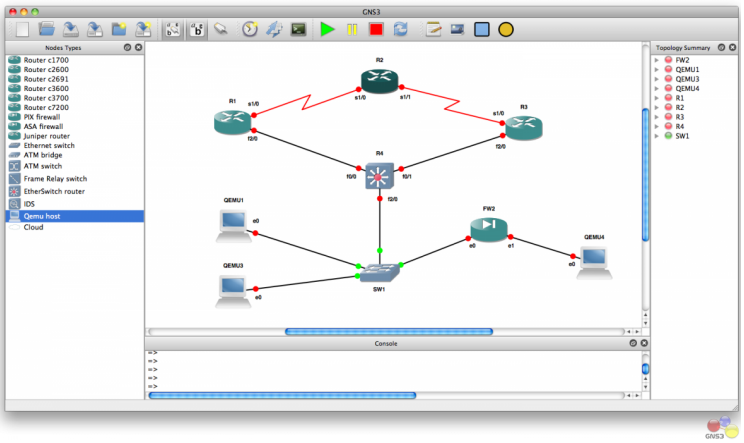
2.2 GNS3

Asi nejslibnějším programem pro simulaci sítí je projekt GNS3 [13]. GNS3 je grafický síťový simulátor, který umožňuje komplexní a věrné simulace, protože je založen na projektech Dynamips[3] [3], Dynagen[4] [14], QEMU[5] [32] a Vir-

[3]Dynamips je projekt pro práci s obrazy Cisco IOS
[4]Dynagen je textový frontend pro Dynamips
[5]QEMU je emulátor procesoru

Obrázek 2.1: NS-3 logo

tualBox[6] [29]. GNS3 se hodí pro studijní (certifikace Cisco CCNA, CCNP, CCIP a CCIE) i výzkumné účely. Tento simulátor umožňuje simulaci těchto prvků: směrovače založené na Cisco IOS, Cisco switche, Cisco PIX firewall, Cisco ASA firewall, Cisco IDS sensors, směrovače Juniper, Linux či Windows PC. Nevýhodou tohoto simulátoru je jeho velká náročnost - uživatel s běžným počítačem může bez problémů simulovat síť o velikosti pouze 10-15 směrovačů (dle výkonnosti hostitelského systému). Pro simulaci síťových prvků od firmy Cisco je nutné dodat vlastní obrazy Cisco IOS - studenti Cisco Academy je mají v ceně, pro školy ale poskytovány zdarma nejsou. Z tohoto důvodu je tento simulátor nepoužitelný pro výukové účely předmětu BI-PSI.

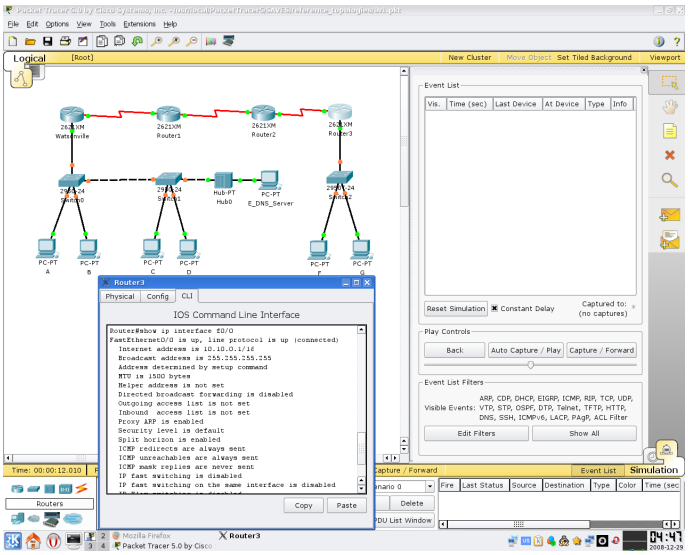


Obrázek 2.2: Ukázka použití GNS3

[6] VirtualBox je multiplatformní virtualizační nástroj od firmy Oracle

2.3 Cisco Packet Tracer

Asi nejznámějším programem na poli simulátorů se směrovači cisco je Cisco Packet Tracer [6] od firmy Cisco Systems. Program umožňuje simulaci síťového provozu na prvcích založených na hardware firmy Cisco Systems. Cisco Packet Tracer umí vizualizovat průchod paketů simulovanou sítí, zvládá konfiguraci síťových prvků v grafickém i textovém módu. Program obsahuje velice málo odchylek od skutečného cisco směrovače. Simulátor je dostupný zdarma pro členy Cisco Networking Academy na platformách Windows a Linux. Škola bohužel nemůže program používat při výuce, protože to jeho licence neumožňuje.

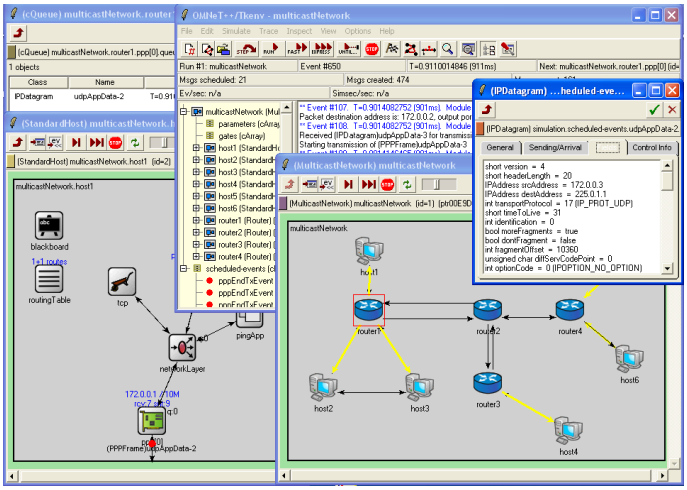


Obrázek 2.3: Cisco Packet Tracer

2.4 OMNeT++ simulátor

Simulační systém OMNeT++ [28] je propracovaný nástroj pro diskrétní simulace. OMNeT++ je postaven na modulární architektuře, proto při správných modulech může simulovat i počítačovou síť. Systém dokáže simulovat Cisco IOS i počítač postavený na linuxu. Tento simulátor umožňuje velmi reálné simulace díky robustně řešenému jádru simulátoru. Aplikace je bohužel nepřehledná a komplikovaná, a proto představu jednoduchého programu pro výuku studentů spíše nesplňuje.

Více se tímto simulátorem zabýval Bc. Jan Michek ve své diplomové práci Emulátor počítačové sítě [18].



Obrázek 2.4: OMNeT++ simulátor

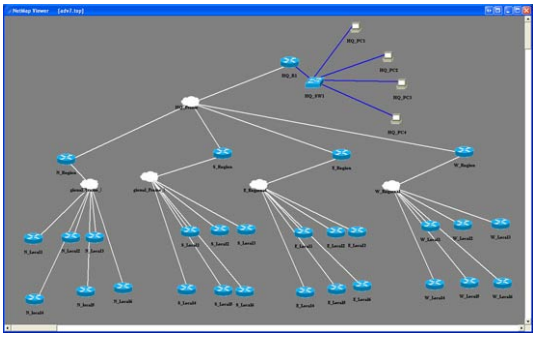
2.5 WebNMS Simulation Toolkit 7

WebNMS Simulation Toolkit 7 [33] je grafický simulátor pro testování a výuku síťových aplikací. Tento nástroj umožňuje simulaci více než 100 000 SNMP služeb (v1, v2c, v3) TL1[7], TFTP[8], FTP[9], Telnet a až 5 000 prvků s Cisco IOS. Simulátor je nabízen pod shareware licencí a je dostupný pro operační systémy Windows, Linux i Unix. Při poskytnutí osobních údajů lze stáhnout plně funkční zkušební verzi na 30 dní. Produkt je nabízen v několika různých verzích, které se liší cenou a počtem dostupných funkcí. Plná časově neomezená verze stojí od $5490[10] pro simulátor s max. 250 prvky s Cisco IOS.

2.6 Boson NetSim Network Simulator

Boson NetSim Network Simulator [1] je simulátor síťového hardware a software a je navržen jako výuková pomůcka pro začínající administrátory směrovačů cisco. Tento simulátor se přímo specializuje na kurzy od firmy Cisco Systems. Aplikace simuluje více než 40 různých síťových prvků od firmy Cisco Systems. Simulátor dále obsahuje grafický i textový konfigurační režim prvků. Simulátor je dostupný pro Windows, Linux i Solaris. Tento software je nabízen ve zkušební verzi na 30 dní. Plná časově neomezená verze stojí od $99 do $349 [11] dle zakoupené verze.

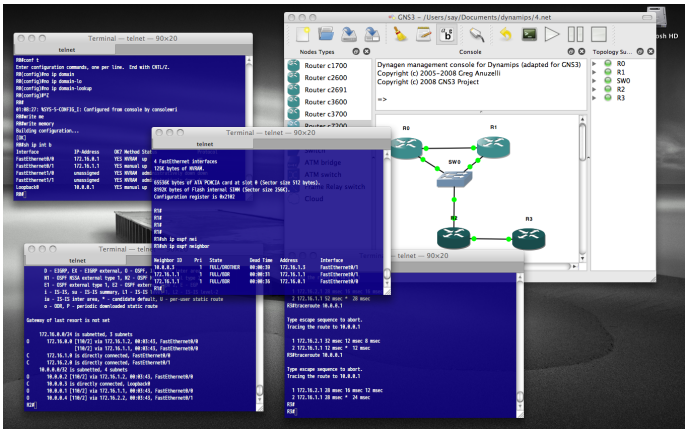
[7] Transaction Language 1
[8] Trivial File Transfer Protocol
[9] File Transfer Protocol
[10] k datu 1.5.2012
[11] k datu 1.5.2012



Obrázek 2.5: Boson NetSim Network Simulator

2.7 Dynamips Cisco 7200 Simulator

Dynamips Cisco 7200 Simulator [3] je simulátor Cisco směrovačů, který je schopen věrných simulací sítě při použití originálního obrazu Cisco IOS. Dynamips je možné spustit na platformách Linux, Mac OS X, Windows. Program umožňuje využívat veškeré funkce Cisco IOS. Software je licencován pod GNU GPL[12], ale obraz Cisco IOS není bohužel volně k dispozici. Proto tento program lze legálně používat jen jako účastník kurzu CNA[13], což studenti obvykle nejsou, proto tento software není vhodný pro výuku. Dynamips je v dnešní době udržován komunitou vývojářů simulátoru GNS3.



Obrázek 2.6: Dynamips Cisco 7200 Simulator

[12] licence pro svobodný software
[13] Cisco Networking Academy

2.8 Závěr

Ani jeden z projektů plně nevyhovuje potřebám předmětu BI-PSI z důvodu přehnané složitosti systému či nevyhovujících licencí pro použití v akademickém prostředí.

KAPITOLA 3

Analýza a návrh

Aktuální verze simulátoru bude v následujících kapitolách identifikovaná názvem **psimulator1** a nová verze jako **psimulator2**. Název vznikl složením zkratky předmětu Y36PSI (Počítačové sítě) a slova simulátor.

3.1 Programovací jazyk a prostředí

Nová verze simulátoru bude implementována v jazyce Java stejně jako aktuální verze. Programy vytvořené v tomto jazyce jsou zpravidla jednoduše přenositelné, čímž se splní jeden z nefunkčních požadavků. Jazyk Java disponuje propracovaným systémem výjimek, proto při neočekávané chybě je možné se zjistit více informací než v jazyce C++ s jeho `Segmentation fault`. Kód napsaný v C++ je možné ladit s debuggerem a zjistit tak řádek s chybou. Díky Javě bude možné získávat chybové výpisy od uživatelů s přesným popisem, kde nastala chyba, aniž by uživatelé používali debugger.

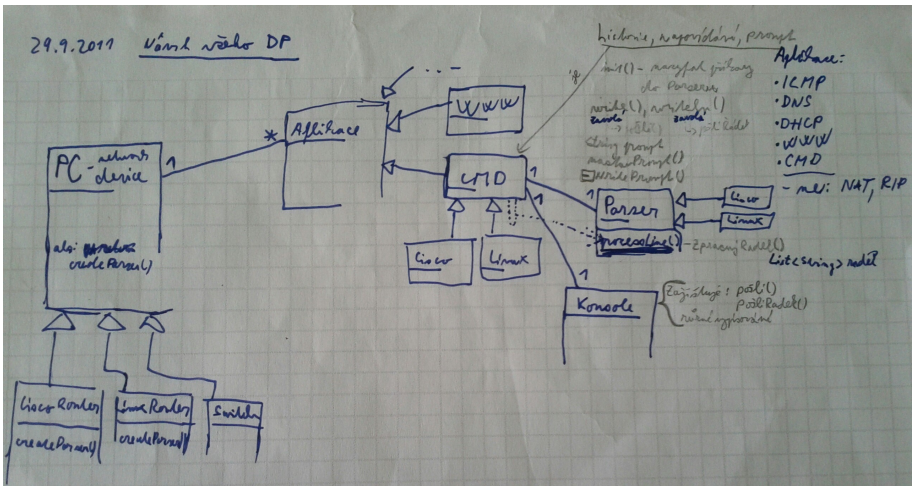
Celá práce bude implementována v prostředí Netbeans IDE[14] verze 7.

3.2 Architektura simulátoru

Jádro psimulator2 jsem navrhoval s Bc. Tomášem Pitřincem, protože se jedná o složitý a rozsáhlý systém, ve kterém může jednotlivec špatně odhadnout a posléze navrhnout důležité komponenty tvořicí jádro simulátoru. Návrhem ve dvojici se tato rizika výrazně minimalizovala. V počátcích návrhu jsme sepisovali požadavky a vymýšleli různá řešení, o kterých jsme dlouho diskutovali a snažili se identifikovat výhody či nevýhody daného řešení. Po zkušenostech z první verze je jasné, že pokud má simulátor alespoň trochu reálně fungovat, tak jeho jednotlivé části musejí fungovat jako nezávislé prvky, které vyřizují

[14] Integrated Development Environment

jim přidělené požadavky. V psimulator1 vše probíhalo v jednom vlákně, proto když chtěl např. příkaz `ping` poslat pět zpráv bez čekání (což je výchozí chování na ciscu), tak se nejdřív musel provést celý první dotaz tzn. dotaz se musel dostat až ke svému cíli, ten musel odpovědět a odpověď musela být vrácena až k tazateli. Pak se teprve mohl poslat další dotaz. Pokud se podíváme na skutečné prvky, které chceme simulovat, tak se jedná vždy o oddělené ekosystémy: hardware: fyzická vrstva (kabely, síťové karty), procesor a software: operační systém, IP stack, aplikace. Tyto všechny části pracují samostatně[15] a nezávisle. Je proto nutné, aby tyto nezávislé části systému pracovaly ve vlastním vlákně. Kvůli snížení zatížení procesoru musí být tato vlákna (= nezávislé části, moduly) uspaná, pokud pro ně není požadavek k vyřízení.



Obrázek 3.1: Návrh architektury jádra simulátoru

Síť bude obsahovat několik typů prvků: směrovač (router), switch a koncové stanice. V budoucnu by mohl být implementován třeba bridge. Nastavení IP adres (a dalších věcí, které budou nastavitelné) bude probíhat přes příkazovou řádku, tak jak je v linuxu a na ciscu zvykem. Konfigurace se bude týkat pouze koncových stanic a směrovačů, protože pouze tam to dává smysl pro potřeby předmětu BI-PSI.

3.2.1 Architektura jádra

Jádro celého simulátoru se bude skládat ze samostatných síťových prvků propojených kabely.

Každý konfigurovatelný síťový prvek bude mít shell (na obrázku 3.1 je to CMD), který bude zajišťovat historii a napovídání příkazů, vypisování

[15] V rámci možností samozřejmě - procesor může počítat pouze jednu úlohu v jeden okamžik.

promptu a dalších informací do konzole klienta. Po připojení nového klienta (po telnetu) se vytvoří nový shell. Ten si vytvoří parser příkazů (podle toho jakého bude prvek typu), který bude zajišťovat zpracování těchto příkazů na konkrétní platformě. Napovídání příkazů bude mít na starosti shell, ale parser mu bude muset nastavit, jaké příkazy podporuje a co by měl napovídat. Dále by měl mít shell konzoli[16], která by měla zajišťovat komunikaci po telnetu s klienty.

Jelikož si tentokrát telnet komunikaci budeme zajišťovat sami, tak by neměl být problém zachytávat stisky klávesy TAB pro napovídání příkazů či šipku nahoru/dolu pro posun v historii příkazů.

Jedním z požadavků je, aby síťový prvek měl podporu pro aplikace, které je možné spustit jako služby (v linuxové terminologii daemony). Tyto aplikace by zajišťovali funkcionality: DNS server, DHCP server a klient, WWW server a pravděpodobně by mohly obsluhovat i ICMP odpovědi pro síťový modul. Aplikace určitě nebudou určeny pro NAT[17] (překlad adres), protože NAT potřebuje být součástí síťové (IP) vrstvy a bude postačující, když bude běžet v jeho vlákně.

Síťový ISO/OSI model bude v tomto simulátoru rozdělen mezi 2 moduly:

1. fyzický modul - bude zajišťovat přenos paketů mezi prvky a přenos paketů do/z síťového modulu

2. síťový modul - bude zajišťovat linkovou, síťovou a transportní vrstvu a bude předávat pakety poslouchajícím aplikacím

Struktura jednoho prvku je na obrázku 3.2.

3.2.2 Fyzický modul

Fyzický modul bude zajišťovat komunikaci v rámci fyzické vrstvy. Bude přijímat pakety od kabelů a ty bude předávat síťovému modulu. Tento modul bude mít na starosti zahazování paketů v případě přeplnění bufferů na rozhraních (síťové karty) a bude také vytvářet zpoždění.

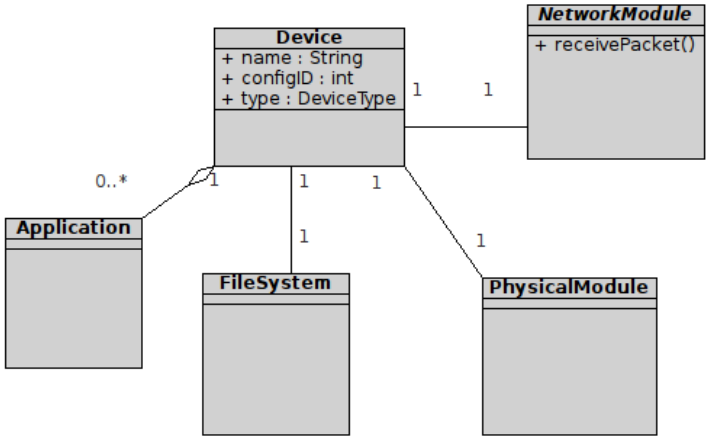
Pro fyzický modul jsem vytvořil dva návrhy:

3.2.2.1 Návrh č.1

Fyzický modul bude přijímat pakety ze shora s odkazem na rozhraní, ze kterého má být daný paket odeslán. Fyzický modul převezme tento paket a přes rozhraní odešle tento paket po kabelu. Druhá strana kabelu si paket přidá do bufferu příchozích paketů a vlákno fyzického modulu předá tento paket svému síťovému modulu.

[16] Teď když píši tuto práci, tak si uvědomuji, že název konzole je špatné a zavádějící pojmenování. Správný název by mohl být např. TelnetManager.

[17] Network Address Translation



Obrázek 3.2: Návrh architektury síťového prvku - Device

Tento modul bude mít k dispozici jedno vlákno pro provádění své práce. Dále každý kabel bude mít své vlákno, které bude předávat pakety z jednoho konce kabelu na druhý a naopak. Vlákna kabelů by pak mohla provádět zpoždění.

3.2.2.2 Návrh č.2

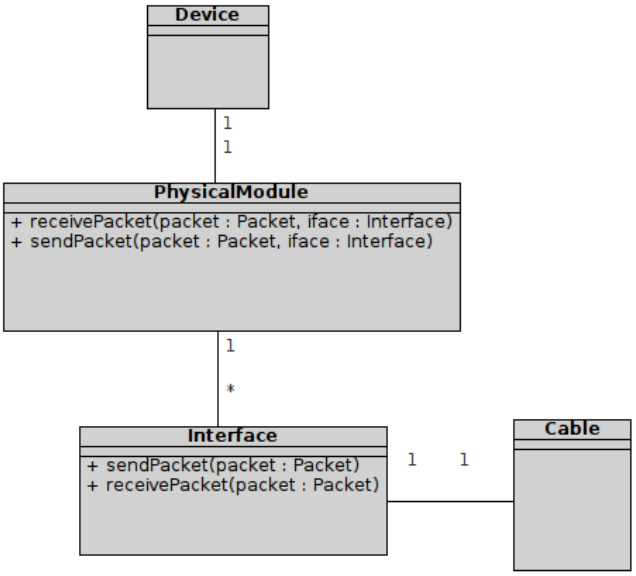
Každé rozhraní bude mít své vlastní vlákno. Když přijde paket po kabelu, tak se uloží na rozhraní (pokud nebude mít plné buffery), pak se probudí jeho vlákno a předá paket síťovému modulu.

3.2.2.3 Diskuze

Druhý návrh je o poznání realističtější, a tak by bylo vhodnější použít tuto variantu. Obávám se však velké náročnosti tohoto řešení, protože 32 portový switch by měl jen pro sebe 32 vláken. Proto volím prozatím návrh č.1. Až bude psimulator2 hotov, tak by se mohla vyzkoušet i druhá varianta a mohla by se provést měření. Pokud by náročnost simulátoru zůstala stejná či podobná, tak by se mohl používat tato verze modulu. Diagram tříd je na obrázku 3.3.

3.2.3 Síťový modul

Síťový modul simulátoru by měl z větší části kopírovat ISO/OSI model. Fyzická vrstva může být mimo tento modul, protože ta bude vždy stejná. Ostatní vrstvy se budou lišit dle typů prvků. ISO/OSI model bude rozdělen na fyzický modul (síťové karty - rozhraní, kabely) a síťový modul jako celek. Síťový modul se bude skládat z 2. až 4. vrstvy, ostatní vrstvy nebudou zatím nutné.



Obrázek 3.3: Třídní diagram fyzického modulu

Na druhé vrstvě bude muset být implementována jednodušší forma ethernetu. Třetí vrstva bude muset implementovat ARP a IP protokol (jejich podmnožinu). Překlad adres bude vsazen přímo do síťového modulu, protože potřebuje měnit části paketů na 3. (IP adresy) i 4. vrstvě (čísla portů). Průchod paketu síťovým modulem je zachycen na obrázku 3.4.

Nový psimulator2 bude mít dvě základní verze síťového modulu:

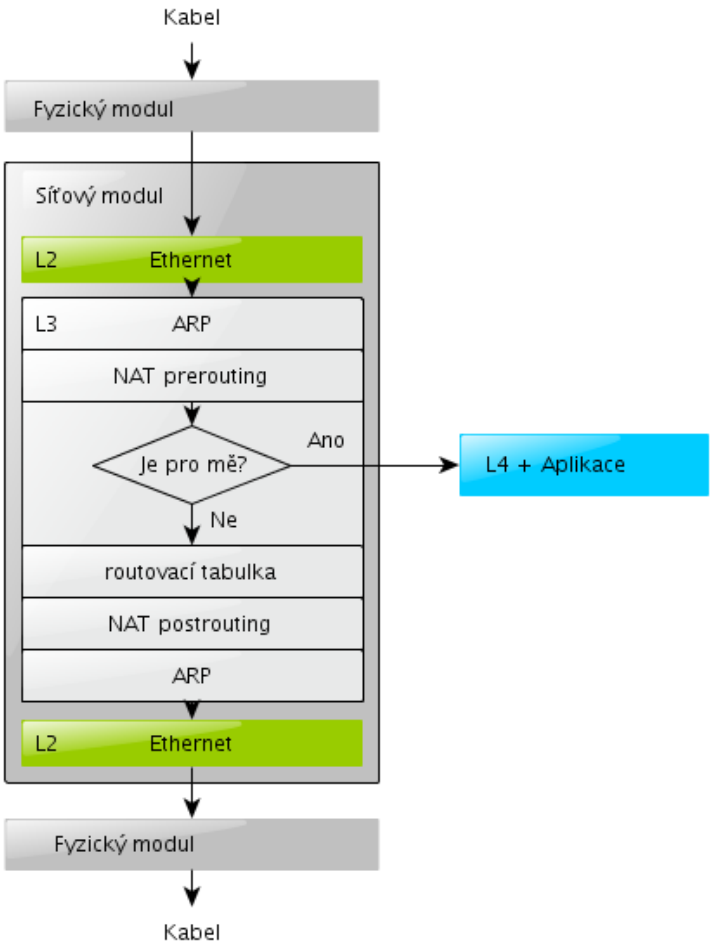
1. SwitchNetworkModule - jednoduchý síťový modul, který bude implementovat pouze linkovou vrstvu (ethernet)

2. IpNetworkModule - složitější síťový modul, který bude rozšiřovat SwitchNetworkModule o síťovou a transportní vrstvu (a případné další vrstvy)

Hierarchii modulů a jeho vrstev popisuje obrázek 3.5.

3.2.3.1 Linková vrstva

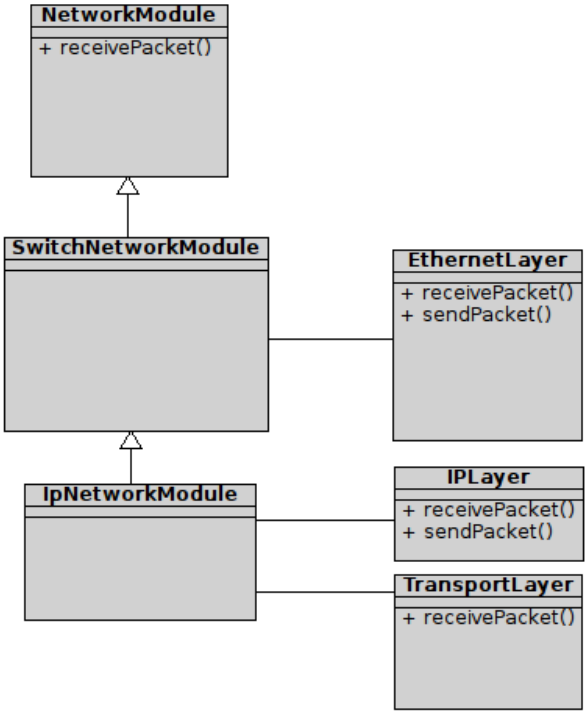
Linková vrstva bude mít k dispozici vlastní vlákno, které bude zpracovávat požadavky (příchozí nebo odchozí pakety) zdola od fyzického modulu a shora od síťové vrstvy. Jejím úkolem bude doručovat pakety adresátům identifikovaných MAC adresami.



Obrázek 3.4: Návrh vrstev síťového modulu

3.2.3.2 Síťová vrstva

Síťová vrstva bude implementovat ARP protokol (viz návrh v kapitole 3.2.5), NAT (překlad adres) a směrování IP paketů (routovací tabulka bude moci být převzata z psimulator1). Tuto vrstvu bude obsluhovat samostatné vlákno. IP část síťové vrstvy se bude výrazně lišit napříč prvky - bude třeba vyčlenit funkčnost do speciálních tříd, které budou zajišťovat odlišné chování. Návrh síťové vrstvy je na obrázku 3.6.



Obrázek 3.5: Třídní diagram síťového modulu

3.2.3.3 Transportní vrstva

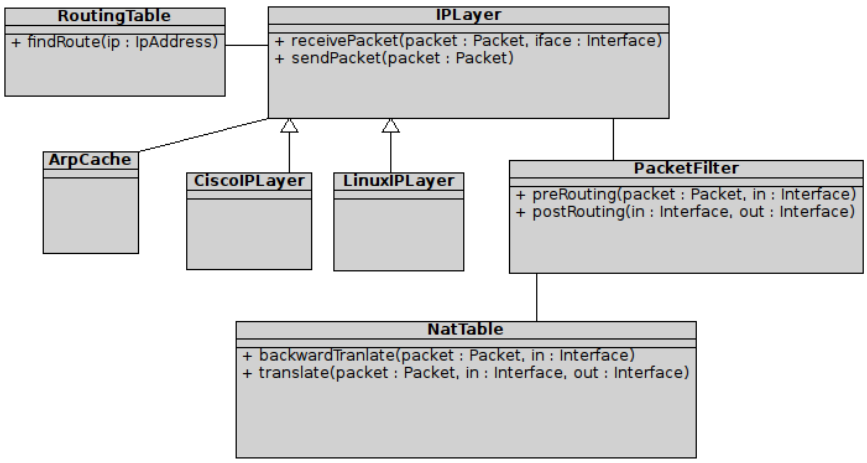
Transportní vrstva bude zajišťovat předávání paketů konkrétním aplikacím. Dále bude registrovat čísla portů, na kterých budou aplikace poslouchat. Vrstva bude v zásadě jednoduchá, nebude obsahovat žádné dlouho trvající operace, proto bude moci pracovat v rámci vlákna síťové vrstvy.

3.2.4 Pracovní vlákno

Z analýzy vyplývá, že bude nutné navrhnout a implementovat formu vlákna, které bude většinu svého života ve stavu `sleeping`. Pokud přijde nový požadavek, tak se změní jeho stav na `running` a požadavek bude vyřízen. Pokud nebude existovat další požadavek, tak se přepne opět do stavu `sleeping`.

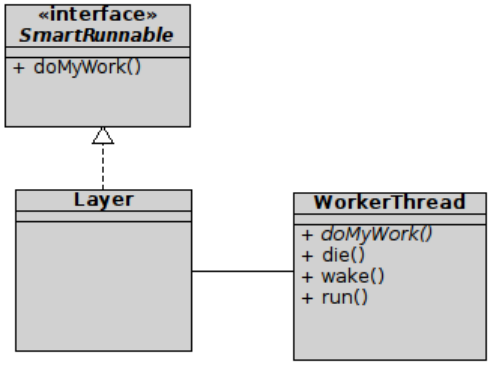
Na tomto principu by měly pracovat tyto komponenty: fyzický modul (kabely), linková vrstva, síťová vrstva a aplikace.

V rámci analýzy jsme s kolegou vytvořili prototyp takového vlákna, který obsluhuje čísla, které mu předává jiná „vrstva“. Zjistili jsme, že náročnost takové implementace je minimální, a proto tento prototyp použijeme na patřičných místech v psimulator2.



Obrázek 3.6: Třídní diagram síťové vrstvy

Návrh takového vlákna je na obrázku 3.7.



Obrázek 3.7: Třídní diagram pracovního vlákna

3.2.5 ARP protokol

ARP (Address Resolution Protocol) je protokol, který se používá v IP protokolu ke zjištění ethernetové MAC adresy sousedního prvku (nexthop) z jeho IP adresy pokud je nutné odeslat IP paket na adresu ve stejné podsíti jako odesílatel. K odeslání paketu přes linkovou vrstvu je nutné znát adresáta linkové vrstvy identifikovaného MAC adresou. Tuto MAC adresu je možné zjistit z IP adresy pmocí ARP protokolu.

Pokud odesílatel chce odeslat paket a nezná sousedovu MAC adresu, tak odešle `ARP request` linkovým broadcastem (ff:ff:ff:ff:ff:ff), což způsobí doručení ARP žádosti na všechny prvky ve stejné síti. Všechny prvky, co dostanou `ARP request`, si uloží informace o odesílateli do své ARP tabulky. Pokud je příjemce zároveň cílem ARP žádosti, tak odešle tazateli `ARP reply` se svojí MAC adresou a IP adresou.

Získané informace se kvůli efektivitě ukládají do ARP tabulky (ARP cache). Záznamy v ARP tabulce vyprší po skončení jejich platnosti. ARP protokol je podrobně definován v RFC 826 [26].

3.2.5.1 Struktura ARP paketu

Struktura ARP paketu je na obrázku 3.8. Pro účely simulátoru bude nutné implementovat pouze podmnožinu položek ARP paketu:

- operation - bitový příznak, zda je paket ARP request (hodnota 1) či ARP reply (hodnota 2).
- sender hardware address - MAC adresa odesílatele ARP paketu
- sender protocol address - IP adresa odesílatele ARP paketu
- target hardware address - MAC adresa cíle ARP paketu
- target protocol address - IP adresa cíle ARP paketu

Příznak `operation` je při zpracování paketu zpracováván až nakonec [20], tedy v době kdy ARP cache už je zaktualizována informacemi z tohoto paketu. Při odesílání paketů u ARP announcements[18] na tomto příznaku nezáleží, pokud jsou ostatní pole správně nastavena.

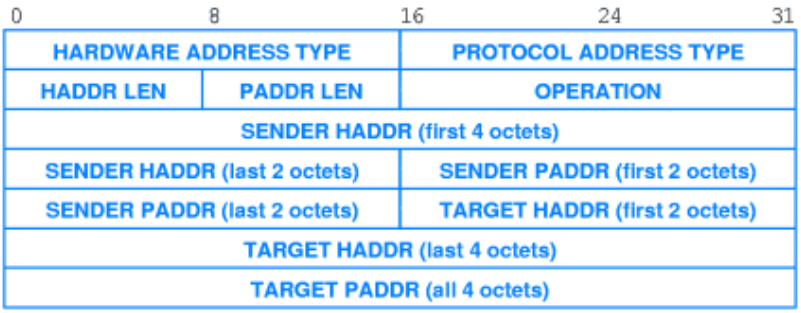
3.2.5.2 Experiment

Pro zjištění a ověření detailů ARP protokolu jsem vytvořil tento experiment: Notebook jsem připojil po kabelu na rozhraní eth0 k domácímu routeru, který pracuje jako switch. Wifi rozhraní notebooku (wlan0) bylo nahozené, nebylo ale připojeno do wifi sítě routeru. Po wifi jsem připojil telefon do sítě a spustil jsem na něm tento příkaz:

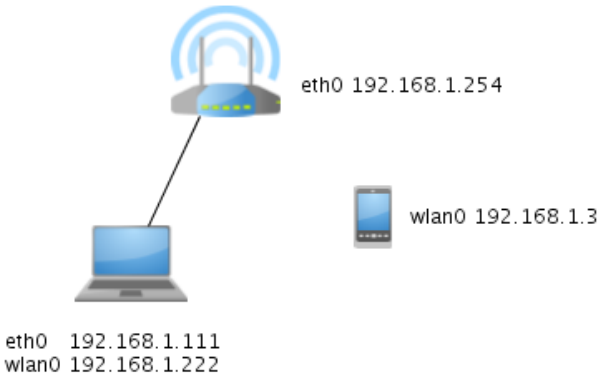
```
telefon~$ ping 192.168.1.222
```

Tento příkaz způsobil odeslání ARP request na linkový broadcast. Notebook obdržel zprávu a jelikož vlastní tuto IP adresu, tak odpověděl ARP reply. Obsahem této odpovědi nebyla MAC adresa rozhraní wlan0 (tedy rozhraní, které má adresu, na kterou byl poslán dotaz), ale MAC adresa rozhraní, ze kterého přišla žádost.

[18] viz kapitola 3.2.5.3 ARP announcements



Obrázek 3.8: Struktura ARP paketu [12]



Obrázek 3.9: Experiment

3.2.5.3 ARP announcements

ARP announcements (oznámení) je speciální typ ARP paketu, který slouží k oznámení vlastní MAC adresy ostatním stanicím na síti. Když si stanice změní adresu, tak může poslat ARP announcements a tím zaktualizovat ARP tabulky ostatních stanic.

Po testech na linuxovém směrovači jsem zjistil, že po změně IP adresy se žádné ARP announcements neposílá. Školní cisca také ARP announcements neposílají. Zatím tato funkcionalita nebude implementována, nicméně v návrhu s tím budu počítat. Pokud bude někdo chtít změnit IP adresu na rozhraní,

tak ji bude striktně měnit pouze a jen přes jednu metodu, do které by mohla být tato funkcionalita přidána.

3.2.5.4 ARP probe

`ARP probe` je `ARP request`, která má nastavenou `sender protocol address` na 0.0.0.0. ARP probe se používá pro detekci kolizí IPv4 adres. Pokud chce někdo implementovat specifikaci RFC 5228 [20], tak musí před nastavením IP adresy zjistit, zda nová adresa již není používána - pošle ARP probe na broadcast.

V našem simulátoru toto nebude implementováno, protože ve školních laboratořích toho není potřeba.

3.2.6 Překlad adres (NAT)

Překlad síťových adres neboli NAT (Network address translation) je způsob jak překládat adresy lokální (vnitřní) sítě na adresu (nebo i na více adres podle druhu a nastavení NATu) veřejnou. Aby překlad mohl fungovat i zpětně, tak si musí prvek, který překlad provádí, zaznamenávat, komu jakou IP adresu přidělil. Pokud se jedná o dynamický překlad adres, tak se někdy překládá několik IP adres na 1 IP adresu. Při dynamickém překladu je nutné přepisovat i zdrojové porty a ukládat si tak dvojice adresa + port.

Překlad adres vznikl kvůli nedostatku veřejných IP adres v roce 1994. V té době sice veřejné adresy ještě nedošly, ale bylo jasné, že ta chvíle brzy nastane. Překlad adres ale není skutečné řešení tohoto problému, nýbrž pouze jeho oddálení. Na jednu IP adresu je možné namapovat maximálně 35535 spojení. Pokud můj notebook při běžném použití obsadí cca 80 portů, tak se může na 1 IP a zapnutý překlad adres schovat přibližně 400 klientů. Pokud si ale jeden klient spustí nějakou P2P [19] aplikaci, tak obsadí 500 portů či více. I z tohoto důvodu nemají poskytovatelé radost, když uživatelé stahují data pomocí P2P sítí.

3.2.6.1 Struktury paketů

Překlad adres v simulátoru se bude týkat převážně ICMP paketů, protože pomocí těchto paketů se posílají dotazy kvůli ověření dosažitelnosti daného stroje v síti. Simulátor ale bude muset podporovat i jiné pakety: TCP, UDP, ARP a ethernet pakety. Ethernet ani ARP pakety se do překladu adres vůbec nedostanou, protože Ethernet bude zpracovávat nejspodnější vrstva (linková) a ARP bude zpracovávat síťová vrstva (kvůli doručení ARP request, více o ARP v kapitole 3.2.5).

Pakety dle vrstev:

1. vrstva 2: Ethernet paket

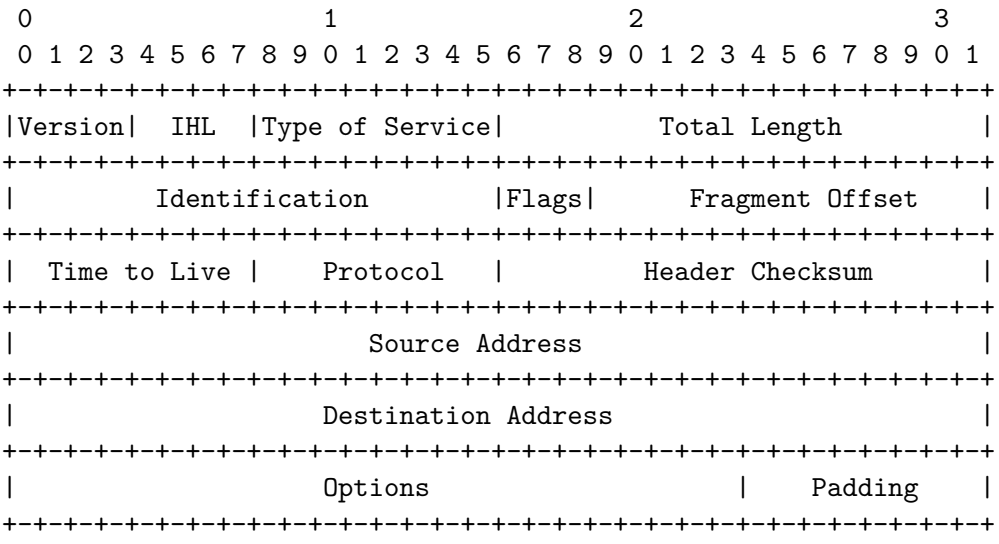
[19] Peer 2 peer

2. vrstva 3: IP paket, ARP paket
3. vrstva 4: TCP paket, UDP paket, ICMP paket

Překladu adres se týkají jen tyto druhy paketů: IP, TCP, UDP a ARP.

3.2.6.1.1 IP paket

Struktura IP paketu je definována dokumentem RFC 791 [22], ukázka z tohoto dokumentu viz níže.

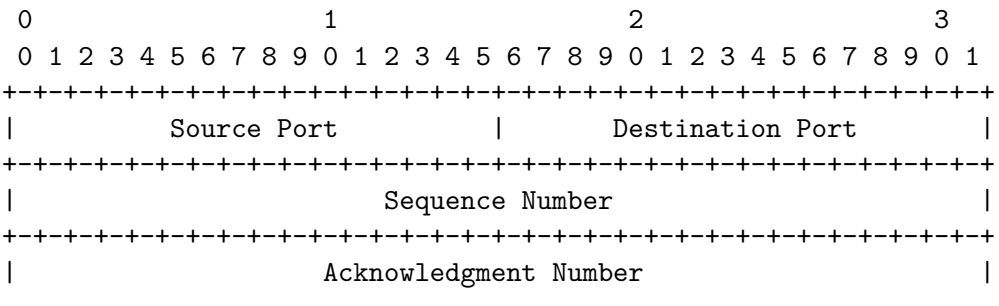


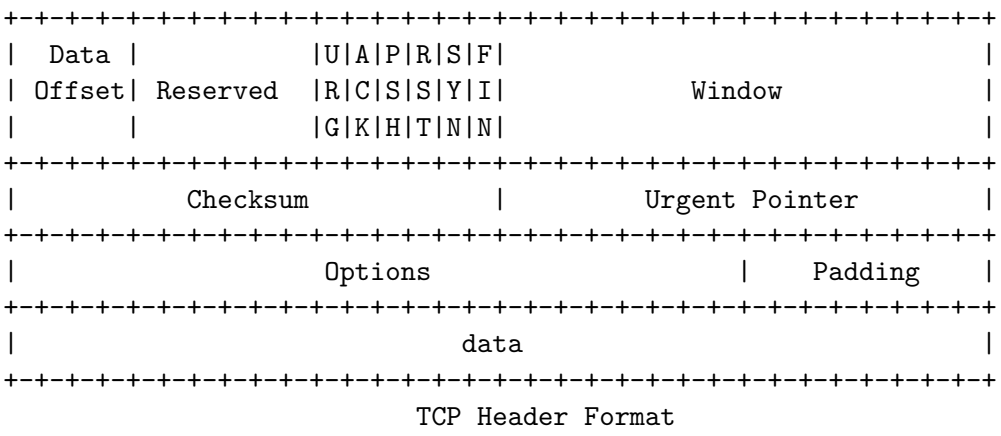
Example Internet Datagram Header

V simulátoru budou potřeba tyto pole: Type of Service, Time To Live, Source Address a Destination Address. NAT bude muset měnit zdrojovou (dopředný překlad) a cílovou (zpětný překlad) adresu paketu. Ostatní položky paketů se budou přidávat při odchodu paketu do skutečné sítě. Kontrolní součty (a další bity) by celý simulátor jen zbytečně zpomalovaly a pro účely této práce by to nepřineslo žádnou hodnotu.

3.2.6.1.2 TCP paket

Struktura TCP paketu je definována dokumentem RFC 793 [25], ukázka z tohoto dokumentu viz níže.



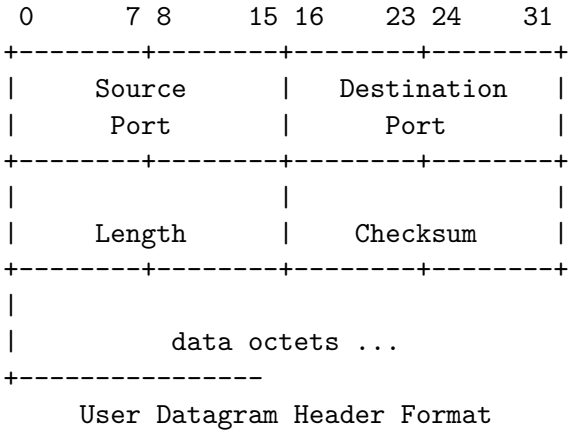


TCP Header Format

Simulátor bude pravděpodobně potřebovat pouze Source Port a Destination Port, protože TCP protokol nebude prozatím implementován. Oba porty se budou přepisovat při překladu adres.

3.2.6.1.3 UDP paket

Struktura UDP paketu je definována dokumentem RFC 768 [21], ukázka z tohoto dokumentu viz níže.

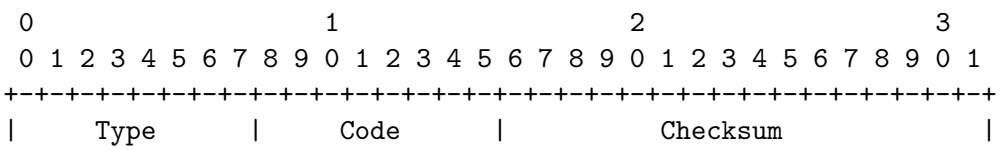


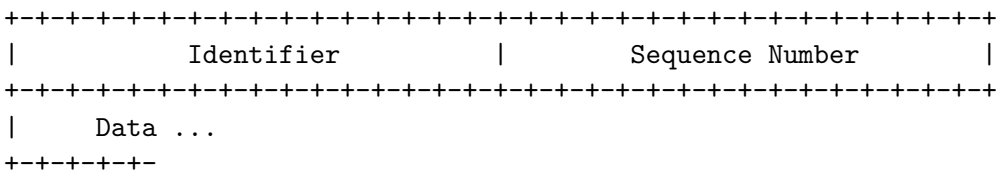
User Datagram Header Format

Stejně jako u TCP paketu bude potřeba v simulátoru jen Source Port a Destination Port.

3.2.6.1.4 ICMP paket

Struktura ICMP paketu je definována dokumentem RFC 792 [23], ukázka z tohoto dokumentu viz níže.





Zde je to zajímavější, protože dle RFC 792 [23] ICMP paket nemá port. V praxi to funguje tak, že když přes NAT prochází ICMP paket, tak se ukládá hodnota Identifier (tu si ukládá NAT tabulka jako port) a přepíše se zdrojová adresa. Když cíl paketu odpoví na dotaz, tak přibalí k odpovědi i původní paket. Překlad adres se podívá do přiloženého paketu (do původní žádosti) a vybere správný záznam z NAT tabulky a přepíše cílovou adresu dle záznamu.

V Type a Code se v simulátoru bude používat tato podmnožina [15]:

- Type 0 - Echo Reply (odpověď na ping)

  Code 0 - No Code

- Type 3 - Destination Unreachable (cíl je nedosažitelný)

  Code 0 - Net Unreachable

  Code 1 - Host Unreachable

  Code 2 - Protocol Unreachable

  Code 3 - Port Unreachable

- Type 4 - Source Quench (fronta na rozhraní je plná)

  Code 0 - No Code

- Type 8 - Echo Request (ping)

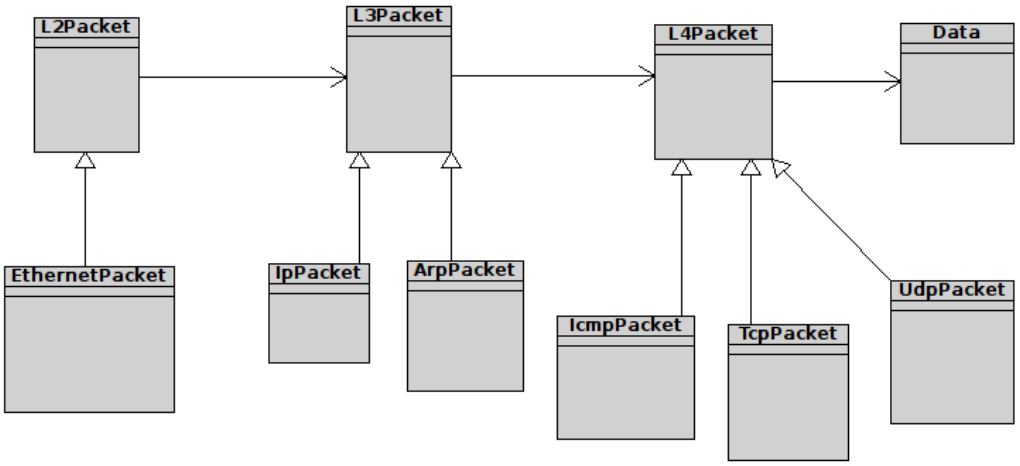
  Code 0 - No Code

- Type 11 - Time Exceeded (zpráva nemohla být doručena)

  Code 0 - Time to Live exceeded in Transit

3.2.6.2 Návrh paketů

V psimulator2 budou 3 základní abstraktní typy paketů: L2Packet, L3Packet a L4Packet. Po kabelech mezi prvky se bude přenášet L2Packet. Ten bude mít odkaz na L3Packet a ten zase bude mít odkaz na L4Packet. L4Packet bude mít pravděpodobně odkaz na nějaká další data. Vše ilustruje obrázek 3.10.



Obrázek 3.10: Třídní diagram struktury paketů

3.2.6.3 Návrh překladu adres

Překlad adres bude z velké části převzat z mé bakalářské práce. Psimulator1 vůbec nepočítá s jiným typem paketů než ICMP. Díky tomu měly ICMP pakety přidánu položku port, ačkoliv skutečný ICMP paket žádnou takovou položku nemá. V psimulator2 bude muset NAT přepisovat položky paketu na 3. i 4. vrstvě stejně jako skutečný překlad adres.

Počet záznamů v překladové tabulce je omezen velikostí paměti DRAM (Dynamic Random Access Memory), jakou router disponuje. Jelikož typický záznam v NAT tabulce má velikost přibližně 160 bytů a dnešní domácí routery mají běžně 4MB DRAM paměť, tak je teoreticky možné uchovávat najednou až 26 214 záznamů, což je více než dost i pro tento simulátor. Proto nebude nutné omezovat ani v psimulator2 velikost NAT tabulky.

Cisco podporuje několik druhů překladů [7] síťových adres:

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

Překlad adres lze u cisca nastavit opravdu detailně, pro tuto práci je však důležitá pouze úzká podmnožina NATu. Detailní popis fungování překladu adres lze nalézt na webových stránkách společnosti Cisco Systems [9].

Více o návrhu k překladu adres bude zmíněno v kapitole implementace 4.4.2.5.

3.2.7 Souborový systém

Psimulator2 bude podporovat souborový systém kvůli konfiguraci a ukládání některých částí na linuxových prvcích. Cílem této analýzy je zjistit, zda je nutné implementovat souborový systém (a příkazy pro něj) pro cisco.

3.2.8 Analýza pro Cisco IOS

Systém souborů je na ciscu podobný jako na linuxu: skládá se z adresářů rozčleněných podle souborových systémů na prefixy, např.:

```
b0902c-esw#show file systems
File Systems:

     Size(b)     Free(b)      Type  Flags  Prefixes
*    7741440      372736     flash     rw   flash:
           -           -    opaque     ro   bs:
       32768       16936     nvram     rw   nvram:
           -           -    opaque     rw   null:
           -           -    opaque     rw   system:
           -           -   network     rw   tftp:
           -           -    opaque     ro   xmodem:
           -           -    opaque     ro   ymodem:
           -           -   network     rw   rcp:
           -           -   network     rw   ftp:
           -           -    opaque     ro   cns:
```

Pro zobrazení všech souborových systémů je v Cisco IOS implementován příkaz `show file systems` (viz výpis výše). Prefixy označují jednotlivé souborové systémy (obdoba linuxových pojmenování /dev/sda).

Cisco IOS dále podporuje příkazy pro práci se souborovým systémem: `cd`, `pwd`, `show <filesystem>`, `delete` atd. Pro účely předmětu BI-PSI by mělo smysl implementovat ukládání a načítání konfigurace. To bude ale řešeno přes interní příkazy[20] simulátoru, jako tomu bylo v psimulator1 z důvodů jednotného ukládání/načítání na všech dostupných prvcích (aktuálně cisco a linux).

Dokumentace k souborovému systému na Cisco IOS je dostupná na stránkách společnosti Cisco Systems [5].

3.2.9 Architektura aplikací

Aplikace budou zajišťovat služby typu DHCP, WWW server, DNS, ale i dlouho trvající příkazy, které by měli pracovat ve vlastním vlákně (ping a traceroute). Aplikace bude nezávislá služba, která bude spouštěna na prvcích se síťovým

[20] V abstraktním parseru příkazů bude příkaz save.

modulem (`IpNetworkModule`). Bude se jednat o obdobu linuxových daemonů. Aplikace budou přijímat pakety ze síťového modulu přes transportní vrstvu. Při svém startu aplikace předá transportní vrstvě port, na kterém hodlá poslouchat. Pro potřeby aplikací ping a traceroute bude existovat možnost startovat aplikaci bez portu s tím, že transportní vrstva jí přidělí volný port, na kterém bude moci poslouchat.

Celý návrh a implementace bude dále v kapitole 4.2.

3.2.10 Načítání a ukládání konfigurace

Ve své bakalářské práci [34] jsem psal vlastní načítání a ukládání celé konfigurace do souboru XML. Ukázalo se to jako vhodné řešení pro ukládání menšího počtu atributů. Výhodou byla absolutní kontrola nad celým procesem načítání a ukládání. Pro potřeby psimulator2 jsem plánoval využití frameworku Castor [2], který umožňuje ukládání javovských objektů přímo do XML a který je dokáže načíst zpět. Velkou výhodou tohoto frameworku je jednoduchost přidání nového atributu do ukládacího procesu. V psimulator1 bylo nutné doimplementovat načítání a ukládání pro každý nový atribut, což je pro potřeby nové verze neudržitelné. V nové verzi je nutné, aby přidání dalšího atributu znamenalo minimální[21] zásah do ukládacího/načítacího procesu.

Po přijmutí dalšího člena do vývojového týmu - Bc. Martina Lukáše je načítání a ukládání jeho úkol a já se tím již dále nebudu zabývat. Pro ušetření práce jsem Martinovi předal svoje poznatky z psimulator1.

3.3 Předpokládaná výpočetní a paměťová náročnost

Předpokládané náročnosti se týkají backendu, protože pouze ten je součástí mé práce.

3.3.1 Výpočetní náročnost

Všechna vlákna v simulátoru budou většinu svého života ve stavu `sleeping`, a tak by simulátor neměl být nijak extrémně náročný. Předpokládám 5% až 15% zatížení procesoru[22] při posílání ICMP request napříč celou sítí se zapnutým překladem adres přes cca 5 síťových prvků. Náročnost samozřejmě vzroste, pokud bude síť obsahovat desítky směrovačů/klientských stanic, které budou generovat síťový provoz. Sítě, na kterých se testují znalosti studentů, ale nebudou představovat vysokou zátěž.

[21] nebo spíše žádný

[22] Žádné speciální nároky na procesor nebudou, simulátor by měl fungovat i na slabším mobilním procesoru např. Intel Atom

3.3.2 Paměťová náročnost

Paměťová náročnost se bude pravděpodobně pohybovat v mezích 40 - 120MB při simulaci sítě o 10 prvcích. Očekávám větší paměťovou náročnost oproti psimulator1 kvůli robustnějším datovým strukturám a přidání nezávislých vrstev. Původní simulátor pracoval přibližně na 5 vláknech při 3 připojených konzolích. Nová verze bude využívat cca 50 vláken (pro menší síť).

3.4 Předpokládaný rozsah práce

Je velice těžké odhadnout počet řádků projektu psimulator2. Psimulator1 má přibližně 12 000 řádek kódu. Psimulator2 (bez frontendu) bude mít určitě přes 30 000, protože v nové verzi půjde o kompletní přepsání architektury simulátoru, přidání úplně nového (mnohem složitějšího) síťového modulu a řady dalších komponent.

KAPITOLA 4

Implementace

V této kapitole se pokusím zmínit všechny důležité části z implementace.

4.1 Parser příkazů

Telnet server pro připojení klientů, shell, historii příkazů a render shellu implementoval kolega Bc. Martin Lukáš. Mým úkolem byla implementace parseru (abstraktního + parseru pro Cisco IOS, linuxový parser implementoval kolega Bc. Tomáš Pitřinec) a jeho napojení na `CommandShell`. Pro nově připojeného klienta se vytvoří parser příkazů (konkrétní dle typu `Device`[23]). Když uživatel odešle příkaz přes telnet, tak ho `CommandShell` doručí parseru metodou `processLine()`.

Čtení příkazů probíhá ve 3 módech:

- COMMAND_READ - výchozí mód, ve kterém se manipuluje s historií (procházení a vkládání příkazů)
- NORMAL_READ - výchozí mód pro dlouhotrvající příkazy[24], ve kterých se neočekává vstup (zde se vůbec nepracuje s historií)
- INPUT_FIELD - mód pro načítání vstupu u dlouhotrvajících příkazů

Implementaci módů měl na starosti kolega, já jsem implementoval jejich ovládání v rámci parseru příkazů.

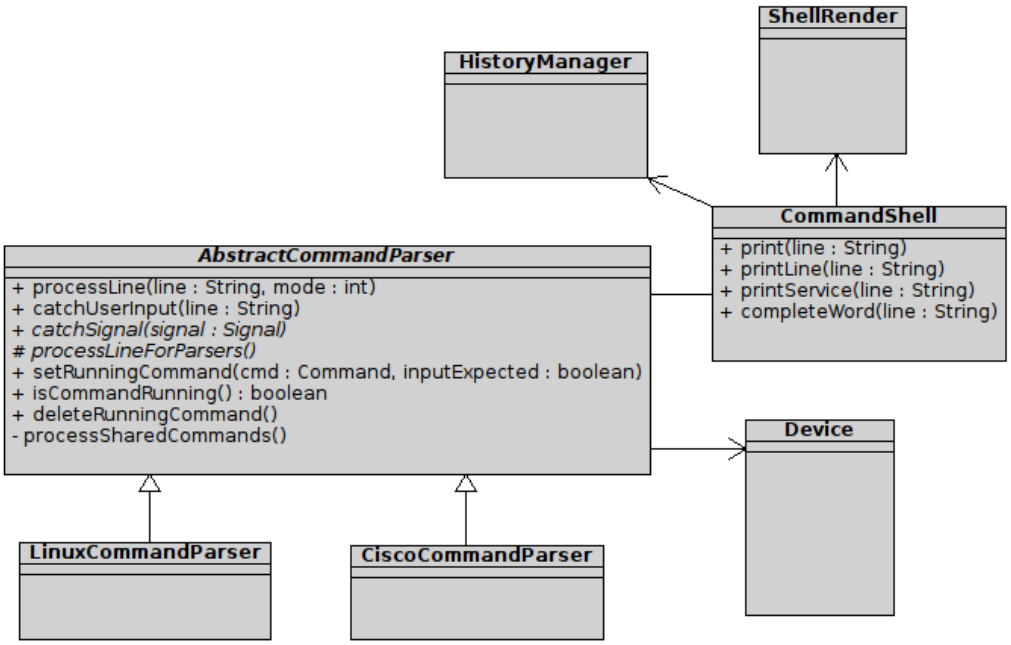
[23] V analýze a návrhu jsem psal o síťových prvcích, v implementaci jsou nazývány jako Device.

[24] Dlouhotrvající příkaz je tu definován jako příkaz, který existuje alespoň tak dlouho, že je schopen zpracovat nějaký další příkaz či vstup od uživatele. Zpravidla se jedná o příkazy, které spustí nějakou aplikaci a čekají na reakci uživatele - ukončení aplikace (typicky příkaz ping).

Pokud parser příkazů vytvoří dlouhotrvající příkaz, tak tento příkaz nastaví v parseru metodou `setRunningCommand()` příznak, zda očekává vstup. V případě očekávání vstupu `CommandShell` volá metodu `catchUserInput()`, která doručí přečtený řetězec dlouhotrvajícímu příkazu. Pokud příkaz vstup neočekává, tak je vložený text vypsán standardně do konzole uživatele (pokud např. u příkazu `ping nix.cz` začnete psát na klávesnici, tak na skutečném linuxu je Váš napsaný vstup tisknut ihned do konzole).

Mód INPUT_FIELD byl zaveden z důvodu potřeby načítat vstup v cisco příkazu `configure`. Když uživatel napíše tento příkaz bez parametrů, tak se parser zeptá uživatele na další konfigurační volbu (očekává hodnoty `terminal`, `memory` nebo `network`). V simulátoru je sice podporována pouze volba `terminal`, nicméně se nabízejí všechny volby pro zachování dojmu skutečného cisca. Při výběru voleb `terminal` nebo `memory` je vypsána zpráva informující uživatele o absenci těchto možností v simulátoru.

Stará verze psimulator1 toto vůbec neumožňovala, proto např. u příkazu `ping` v Cisco IOS nebylo možné vkládat postupně parametry tohoto příkazu[25]. Na konci života dlouhotrvajícího příkazu je vždy zavolána metoda `deleteRunningCommand()`, která odregistruje dlouhotrvající příkaz z parseru a která nastaví zpět COMMAND_READ mód `CommandShell`.



Obrázek 4.1: Třídní diagram parseru příkazů

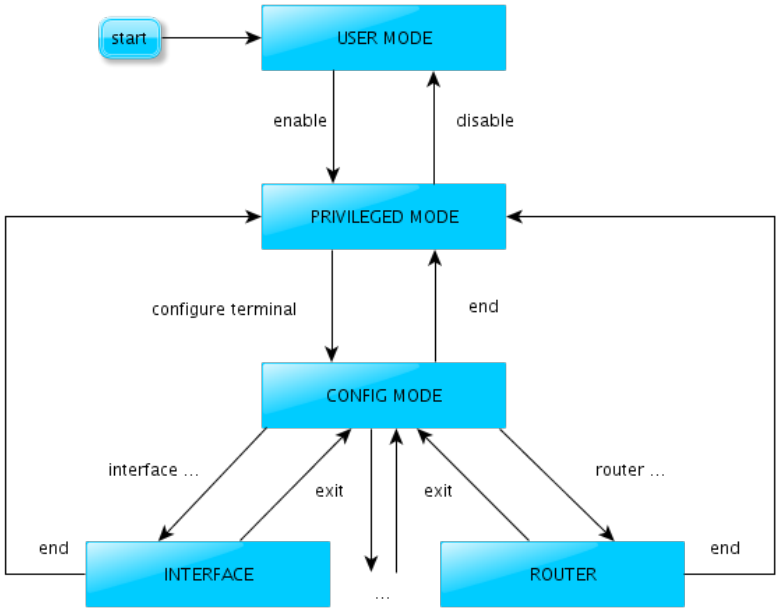
[25]Tato funkcionalita není implementována, nicméně simulátor je na ni plně připraven a je to pouze záležitost parsovací funkce u příkazu PingCommand.

Struktura parseru příkazů a shellu je na obrázku 4.1.

4.1.1 Doplňování příkazů pro Cisco IOS

Doplňování příkazů v psimulator1 nabízelo „vždy všechno“, protože doplňování zajišťoval externí program `rlwrap`. Implementace vlastního systému doplňování příkazů ale nebyla možná, protože telnet server dostával vždy jen celý řádek (a stisknutí klávesy TAB se nikdy neposílalo). Díky tomu, že kolega integroval do psimulator2 telnet server, který umožňuje komunikaci po znaku[26], bylo možné implementovat vlastní doplňování příkazů.

Každý `Device` vlastní sadu tzv. „doplňovačů“ v kódu jsou pod jménem `Completer`. Tuto sadu jsem implementoval jako hash mapu, kde klíčem je číslo módu parseru[27] a hodnotou je `Completer`, protože doplňování příkazů se liší dle aktivního módu parseru.



Obrázek 4.2: Přehled základních módů Cisco IOS

Na obrázku 4.2 jsou popsány základní módy Cisco IOS, které jsem implementoval (kromě módu `ROUTER`) v první verzi simulátoru. Pro potřeby nové verze jsem přidal další příkazy (více viz 4.1.2) a celkově vylepšil části některých příkazů. Cisco doplňuje pouze rozepsané části příkazů a jen v případě, že je možné jednoznačně doplnit příkaz např. `show run` doplní na

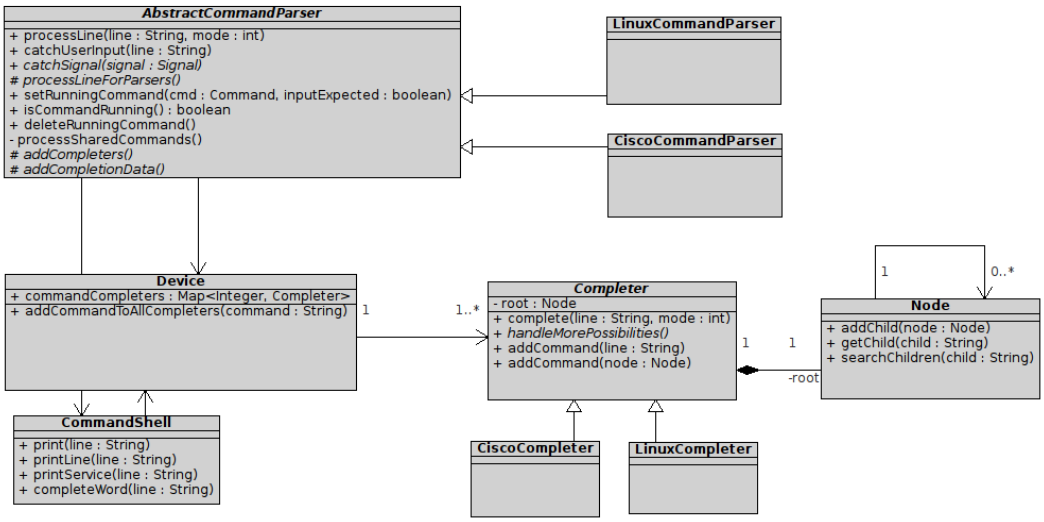
[26] Každý napsaný znak se ihned odesílá od klienta na server.
[27] Mód parseru neplést se čtecím módem shellu.

`show running-config`. Pokud je možností pro doplnění více, tak klávesa TAB nic nedoplní ani nic nevypíše[28]. Cisco IOS používá pro zobrazení dostupných příkazů znak '`?`', který způsobí okamžité vypsání seznamu podporovaných příkazů.

Všechny dostupné módy pro skutečný Cisco IOS jsou popsány v dokumentaci na stránkách společnosti [4].

Reakci na '`?`' jsem zatím nestihl implementovat, nicméně v budoucnu by to neměl být problém, protože '`?`' by mohl čerpat z `Completer`, který si drží seznam podporovaných příkazů. Z psimulator1 jsem převzal příkaz `help`, který umožňuje vypsání seznamu podporovaných příkazů, proto z tohoto důvodu měla implementace '`?`' nízkou prioritu.

Každý konkrétní parser příkazů si při svém startu kontroluje, zda již má jeho `Device` vyplněné příkazy pro doplňování. Pokud dostupné `Completery` nemají vyplněný strom příkazů, tak jsou ihned vyplněny. Doplnitelné příkazy si drží `Device` a ne parser či shell, protože by každé připojení klienta znamenalo znovuvyplnění tohoto stromu příkazů.



Obrázek 4.3: Třídní diagram doplňování příkazů - Completer

Na obrázku 4.3 je UML diagram zobrazující vztahy mezi třídami, které jsou relevatní pro doplňování příkazů. Jakmile klient stiskne klávesu TAB, tak `CommandShell` zavolá ve své metodě `completeWord()` na aktuálně používaném `Completeru` metodu `completeLine()`. Pokud tato metoda vrátí neprázdný řetězec, tak je jasné, že se podařilo doplnit řádek na nějaký příkaz a ten je vypsán do klientské konzole. Pokud je dostupných možností více, tak

[28] Linuxová příkazová řádka naopak možnosti pro doplnění vypíše.

`Completer` zavolá abstraktní metodu `handleMorePossibilities()`, která zajistí rozdílné chování v linux a Cisco IOS.

4.1.2 Nové příkazy pro Cisco IOS

Pro potřeby psimulator2 jsem implementoval několik nových příkazů, které ve staré verzi simulátoru nebyly obsaženy:

- show arp
- show ip interface
- show ip interface brief

4.1.2.1 Příkaz show arp

Při implementaci tohoto příkazu jsem čerpal z oficiální dokumentace [8] na stránkách Cisco Systems. Takto vypadá výpis ze skutečného cisco switche:[29]

```
b0902c-esw#show arp
Protocol  Address    Age (min)  Hardware Addr   Type   Interface
Internet  10.0.0.1           0   6c50.4dae.ac00  ARPA   Vlan201
Internet  10.9.120.3         -   0012.d97c.10c0  ARPA   Vlan201
```

Příkaz vypíše všechny (dynamické i statické) záznamy přítomné v ARP cache (více o ARP cache v kapitole 4.4.2.3) ve formátu:

- jméno protokolu - v simulátoru pouze „Internet“
- IP adresa prvku
- stáří ARP záznamu v minutách
- MAC adresa prvku
- typ záznamu - dynamický nebo statický záznam (klíčová slova Dynamic a Static), tento switch zjevně místo Dynamic vypisuje jméno protokolu, kterým byl daný záznam zjištěn)
- rozhraní, ze kterého je daný záznam dosažitelný

Odchýlil jsem se od skutečného cisca v poli stáří ARP záznamu, protože by bylo málo patrné, kdy který záznam vyprší. V případě nutnosti lze toto chování jednoduše změnit.

[29] Jedná se o výpis switche na strahovských kolejích z bloku 9, 3. switch na 2. patře.

4.1.2.2 Příkaz show ip interface

Tento příkaz (a jeho zkrácená verze `brief`) jsem přidal kvůli poznatkům z uživatelského testování, protože si někteří testeři stěžovali, že nevědí, jak zjistit stav rozhraní. Všechny potřebné informace se dali vyčíst z příkazu `show running-config`, ale stejně jsem implementoval tyto příkazy především kvůli jasnému výstupu u příkazu `show ip interface brief`:

```
cisco1#show ip interface brief
Interface        Ip-Address  OK? Method Status Protocol
FastEthernet0/0  192.168.2.2 YES manual up     *not implemented*
FastEthernet0/1  10.0.0.1    YES manual up     *not implemented*
```

Výpis pochází z psimulator2 a konfigurace `laborka.xml` - prvek cisco1.

Výpis asi není nutné nijak komentovat, kromě sloupce `Protocol`, který značí, zda je linka na tomto rozhraní funkční. V terminologii Cisco to znamená, že dané rozhraní přijímá `keepalive` pakety oběma směry. Psimulator2 žádné takové pakety zatím neposílá. Z tohoto důvodu vypisuji pro tyto dva příkazy hlášku o neúplné implementaci „*not implemented*“.

Při implementaci těchto příkazů jsem čerpal z oficiální dokumentace [10] na stránkách Cisco Systems.

4.2 Aplikace

4.2.1 Návrh aplikace

Každá aplikace v psimulator2 přijímá pakety ze síťového modulu konkrétně z transportní vrstvy. Aby mohla aplikace přijímat a zpracovávat došlé pakety, tak jsem využil koncept pracovního vlákna `WorkerThread`, o kterém jsem psal v kapitole 3.2.4. Aplikace obsahuje metodu `doMyWork()`, která je abstraktní, takže ji implementuje až konkrétní aplikace. Myšlenka je taková, že po startu (a nějaké případné činnosti) se aplikace uspí a bude čekat, až ji někdo probudí - v případě aplikací ji bude budit příchozí paket ze síťového modulu. V tom okamžiku se aplikace probudí a zpracuje všechny přijaté pakety v metodě `doMyWork()`.

Spouštění aplikací bude ve většině případů zajišťovat parser příkazů, protože dostává povely od uživatele. Aplikace bude obsahovat tyto metody:

- receivePacket - bude přijímat pakety ze síťového modulu, po přijetí paketu probudí `WorkerThread`, který se postará o doručenou zásilku v metodě `doMyWork()` u aplikace
- start - provede všechny potřebné operace nutné ke spuštění aplikace
- exit - provede všechny potřebné operace nutné k ukončení aplikace
- kill - ukončí aplikaci bez ukončovacích výpisů

- atStart - abstraktní metoda, která bude sloužit k implementaci výpisů (akcí, kontrol) při startu aplikace
- atExit - abstraktní metoda, která bude sloužit k provedení akcí při ukončování aplikace
- atKill - abstraktní metoda, která bude sloužit k provedení akce při násilném (předčasném) ukončení aplikace
- doMyWork - abstraktní metoda, která bude převážně sloužit ke zpracování došlých požadavků

Aplikace si při startu bude muset zaregistrovat port ve transportní vrstvě, aby jí mohla předávat pakety cílené na tento port. Dále se aplikace při startu zaregistruje u svého prvku - `Device` pod svým PID (Process ID), aby samotný `Device` měl přehled, co je u něho spouštěno a aby mohl své aplikace ukončit v případě potřeby.

4.2.2 Implementace aplikace

Diagram implementovaných tříd je na obrázku 4.4.

Při startu aplikace se nastaví příznak `running`, vytvoří se nový `WorkerThread` (= nastartování pracovního vlákna) a aplikace se zaregistruje nejdříve u `Device` a poté i v transportní vrstvě. Vše ilustruje následující výpis kódu ze třídy `Application`: [30]

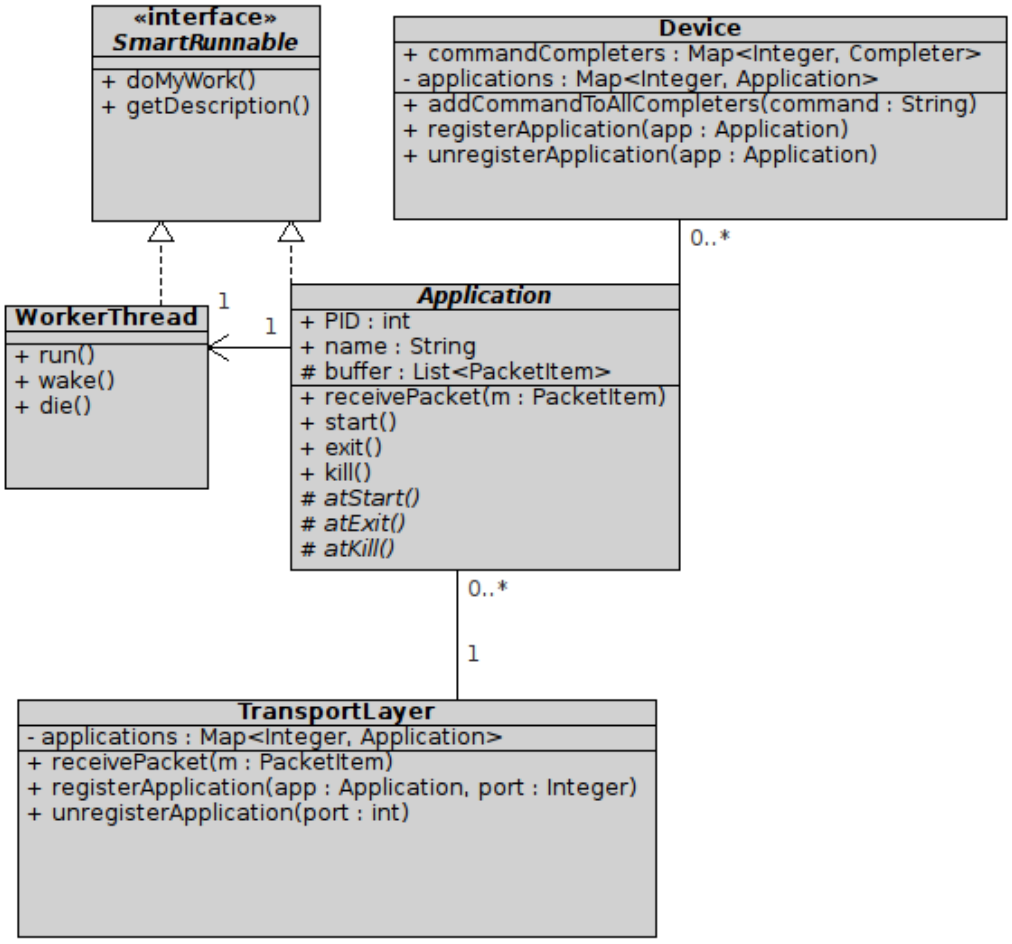
```
1 public synchronized void start() {
2     if (!running) {
3         running = true;
4         this.worker = new WorkerThread(this);
5         device.registerApplication(this);
6         this.port = transportLayer.registerApplication(this, port);
7         atStart();
8     }
9 }
```

Obdobným způsobem jsou implementovány i metody `exit` a `kill`.

V první verzi implementace aplikace přijímala od transportní vrstvy pouze pakety, což se později ukázalo jako nedostatečné, protože některé aplikace potřebují např. informace o rozhraní, na které byl paket doručen.

Transportní vrstva předá příchozí paket (a jiné další informace) přes metodu `receivePacket()`. Tato metoda přidá přijaté informace do bufferu a probudí pracovní vlákno, které se už postará o zbytek. Takto to vypadá v kódu:

[30] Výpis je pro přehlednost zkrácen o logovací zprávy.



Obrázek 4.4: Třídní diagram aplikací

```
1 public void receivePacket(PacketItem packetItem) {
2     if (running) {
3         // log incomming packet
4         buffer.add(packetItem);
5         worker.wake();
6     } else {
7         // log orphan packet
8     }
9 }
```

4.2.3 Návod na přidání nové aplikace

Aplikace je možné jednoduše rozšířit o tyto dva základní druhy aplikací:

1. jednovláknové aplikace, které pouze poslouchají provoz na nějakém portu
2. dvou-vláknové aplikace, které rozesílají pakety po síti a zároveň je přijímají

Tyto dva druhy aplikací se liší pouze v počtu vláken, která mají k dispozici. Vytvoření nové aplikace obou typů je téměř stejné, proto popíši tu dvou-vláknovou verzi.

4.2.3.1 Vytvoření dvou-vláknové aplikace

Nejdříve je nutné si vytvořit třídu reprezentující novou aplikaci. Tato třída bude rozšiřovat `TwoThreadApplication`, což znamená, že bude nutné implementovat metody `doMyWork()` a `run()`. První zmíněná metoda je určena pro zpracování přijatých paketů od síťového modulu a druhá může ve stejném čase provádět libovolnou činnost. Dále bude nutné implementovat metody `atStart()`, `atExit()` a `atKill()`, jejichž účel je popsán v kapitole 4.2.1.

Nová aplikace si může nastavit své jméno a port, na kterém bude poslouchat. Jakmile přijde aplikaci nějaký paket, tak je probuzen `WorkerThread`, který má za úkol zpracování přijatých paketů, a je spuštěna metoda `doMyWork()`. Pro odeslání nového paketu je možné použít transportní vrstvu, kterou vlastní abstraktní aplikace, a tak je dostupná i v nové třídě. ICMP pakety lze odesílat přes `Icmphandler` ve transportní vrstvě.

4.2.4 Ping

Program ping je užitečný nástroj pro zjišťování dostupnosti vzdálených strojů připojených do sítě. Program odesílá `ICMP request` (žádosti) a v případě úspěchu očekává `ICMP reply`. Ping jsem implementoval následovně.

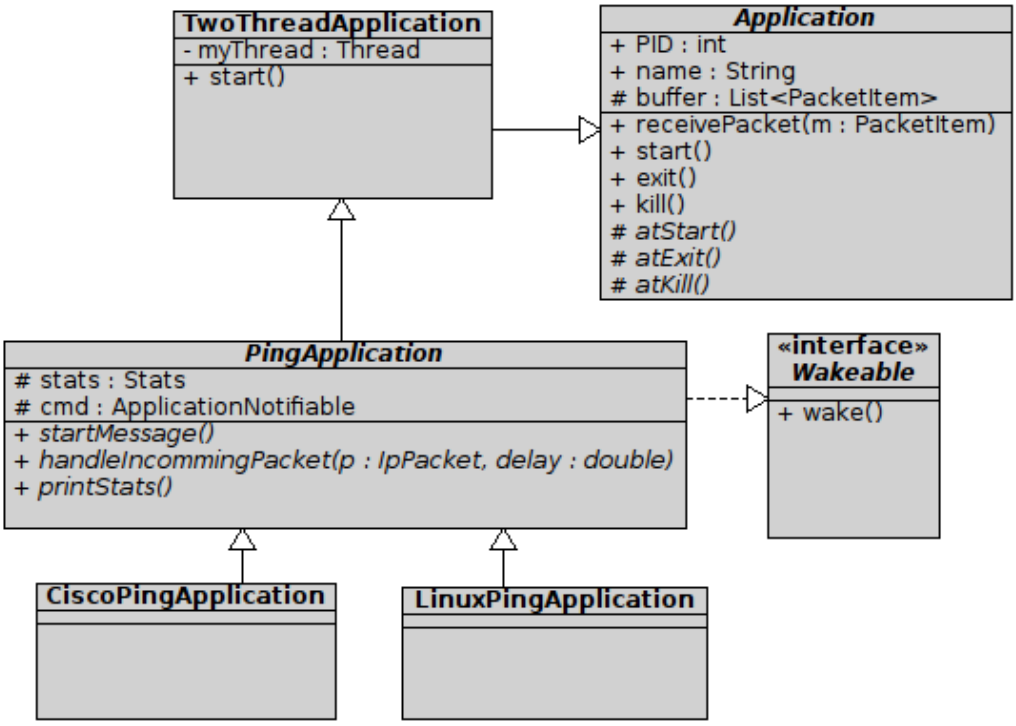
Když parser příkazů obdrží příkaz `ping`, tak mu předá řízení. Příkaz nejdříve zpracuje předaný řádek a pokud je vše v pořádku, tak nastartuje svoji aplikaci - `CiscoPingApplication` a `LinuxPingApplication` dle platformy.

Po startu ping aplikace se okamžitě začnou posílat pakety (`ICMP request`) dokud uživatel nepřeruší odesílání paketů (linux) nebo dokud se neodešle nastavený počet žádostí (cisco).

Nejprve jsem implementoval verzi, ve které se odesílaly `ICMP request` ve vlákně parseru a příchozí pakety se zpracovávaly ve vlákně ping aplikace. Toto řešení se ukázalo sice jako funkční, ale z hlediska návrhu simulátoru jako nevhodné řešení, protože parser má sloužit jako spouštěč příkazů a aplikací a ne jako pracovní vlákno jiných aplikací. Tato první verze by navíc nevyhovovala konceptu signálů `Ctrl+C` pro ukončování aplikací/příkazů.

Pro zlepšení situace se posílání paketů přesunulo do vlastního vlákna, které je spuštěno ihned po startu aplikace. Pro tyto případy vyvinul kolega Bc. Tomáš Pitřinec třídu `TwoThreadApplication`, která tuto funkčnost přidává.

Třídní diagram přesně popisující strukturu je na obrázku 4.5.



Obrázek 4.5: Třídní diagram PingApplication

4.2.5 Traceroute

4.2.5.1 Návrh

Program traceroute slouží ke zjištění posloupnosti směrovačů, kterými paket musí projít, aby se dostal ke svému cíli. Traceroute pracuje na principu posílání paketů s hodnotou TTL[31], kterou postupně inkrementuje z hodnoty 1 až do dosažení cíle či maximálního nastaveného TTL.

Program traceroute je dostupný v několika různých implementacích:

- Windows tracert - posílá ve výchozím nastavení ICMP pakety, chování lze změnit pomocí parametru
- linux/unix traceroute - posílá ve výchozím nastavení UDP pakety, chování lze změnit pomocí parametru
- tcptraceroute [17] - odesílá TCP pakety - nebudu se jím dále zabývat, protože se nejedná o standardní nástroj

[31] Time To Live

Posílání ICMP paketů

Odesílají se klasické `ICMP request` pakety a je očekávána odpověď `Time To Live Exceeded` nebo nic (v případě že nemohla být doručena odpověď zpět odesílateli). V případě dosažení cíle je očekávána `ICMP reply`.

Posílání UDP paketů

Odesílají se UDP pakety na porty 33434 - 33534 (tyto porty musejí být pro správnou funkčnost povoleny ve firewall všech směrovačů) a je očekávána zpráva `Time To Live Exceeded` nebo nic. V případě dosažení cíle je očekávána zpráva `Destination Port Unreachable`, protože na cílovém prvku žádná aplikace na těchto portech zpravidla neposlouchá. Pro případ, že by někdo na takovém portu shodou okolností poslouchal, program traceroute posílá více paketů se stejnou hodnotou TTL vždy na jiný port, aby alespoň na nějaký paket mohla být odeslána očekávaná zpráva `Destination Port Unreachable`.

Mnoho serverů (mezi českými např. nix.cz, seznam.cz) na UDP pakety od programu traceroute vůbec neodpovídá (servery asi nemají pravidla ve firewall), proto jsem se rozhodl implementovat variantu s posíláním ICMP paketů.

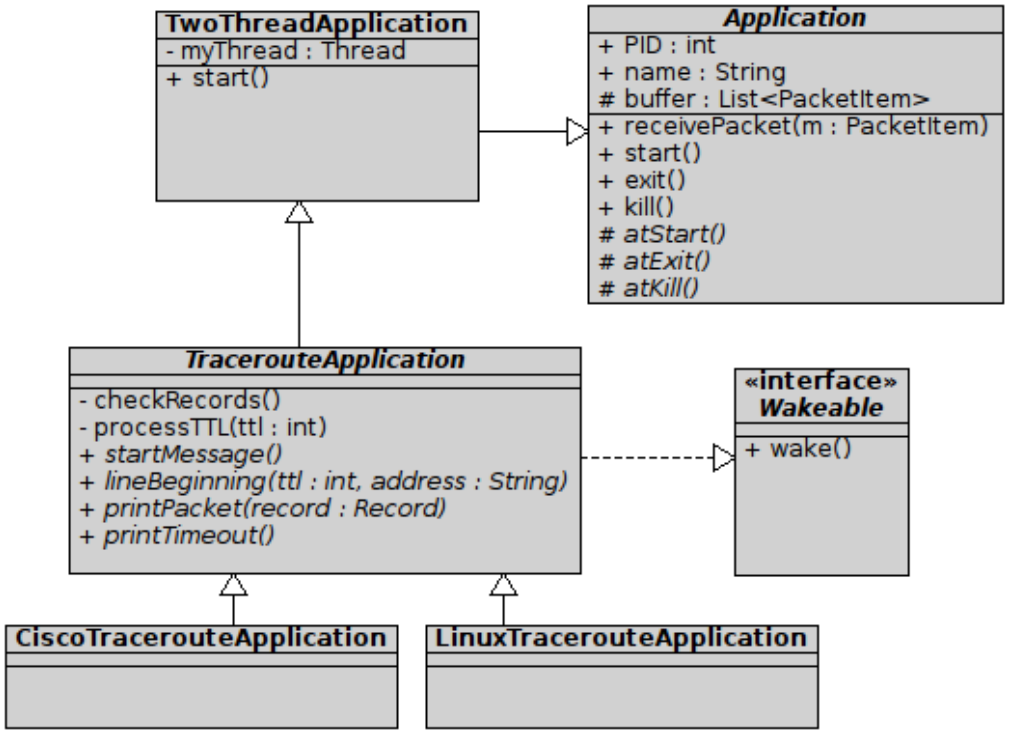
4.2.5.2 Implementace

Implementace je ve své podstatě velice podobná jako u `PingApplication`, převážně se jedná o odlišné zpracování přišlých paketů. Linuxová traceroute aplikace ve výchozím stavu odesílá 3 žádosti pro každé TTL. Já vypisuji řádek s daným TTL buď v době, kdy přišly všechny odpovědi pro dané TTL a nebo pokud vypršel timeout pro poslední paket s daným TTL. Vypisování TTL řádků zajišťuje metoda `checkRecords()`, která kontroluje, co kdy vypršelo a co už (ne)přišlo.

Podrobný třídní diagram popisující strukturu traceroute aplikace je na obrázku 4.6.

Vypisování přijatých paketů pro cisco jsem implementoval dle dokumentace k příkazu traceroute pro Cisco IOS [11]:

```
! -- success
* -- time out
N -- network unreachable
H -- host unreachable
P -- protocol unreachable
A -- admin denied
Q -- source quench received (congestion)
? -- unknown (any other ICMP message)
```



Obrázek 4.6: Třídní diagram TracerouteApplication

4.3 Fyzický modul

Nejprve jsem implementoval první verzi návrhu 3.2.2.1. Simulátor fungoval dobře, ale nebylo možné simulovat alespoň základní zpoždění na rozhraních. První návrh vytvářel zpoždění pouze na kabelu a nebylo možné přidat zpoždění zpracování paketů na rozhraních. Po dokončení většiny částí simulátoru jsem implementoval druhou verzi síťového modulu 3.2.2.2, kde jsou tyto výpočty zohledněny.

V nové verzi fyzického modulu (pojmenovaného jako `PhysicalModuleV2`) je většina práce přesunuta na rozhraní z 1. vrstvy ISO/OSI modelu (zde `Switchport`). Samotný fyzický modul funguje jako fasáda mezi switchporty a síťovým modulem.

Průchod paketů fyzickým modulem od kabelů je následující:

1. paket je doručen na switchport po kabelu, pokud má switchport plnou frontu, tak daný paket zahodí a pokusí se odeslat zprávu ICMP Source Quench[32]

[32]ICMP Source Quench je odesílán pokud má síťový prvek plné příchozí buffery [24].

2. switchport je probuzen příchozím požadavkem a začne vyřizovat všechny pakety, které se nacházejí v bufferech k přijetí (`receiveBuffer`) i k odeslání (`sendBuffer`)

3. paket k přijetí je přes fasádu (`PhysicalModuleV2`) předán síťovému modulu

Průchod paketů fyzickým modulem od síťového modulu je následující:

1. paket je doručen fyzickému modulu, který ihned předá tento paket příslušnému switchportu

2. switchport je probuzen příchozím požadavkem a začne vyřizovat všechny pakety

3. paket k odeslání je předán do odesílacího procesu:

   a) pokud ke switchportu není připojen kabel, tak je vyřizovaný paket zahozen

   b) pokud není druhý konec kabelu připojen do nějakého switchportu, tak je vyřizovaný paket zahozen

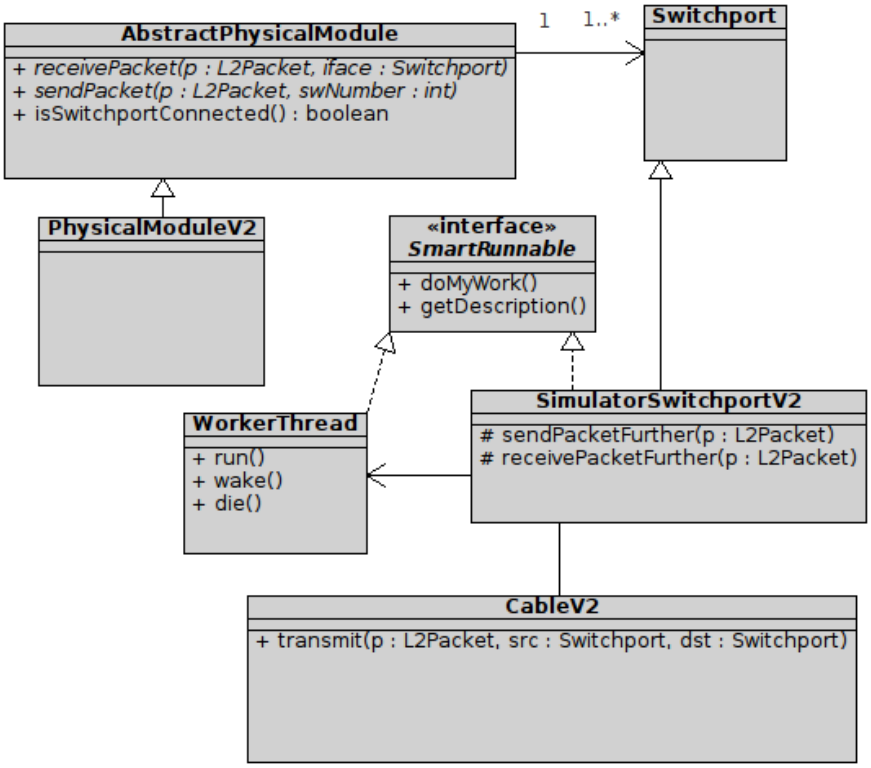
   c) pokud je vše v pořádku, tak se vytvoří zpoždění při odesílání a zpracování paketu a paket je následně odeslán na druhou stranu kabelu

Zpoždění kabelu se vypočítává dle rychlosti sítě podle následujícího vzorce:

$$\frac{n * 8}{m * 1024^2}$$

n .. velikost paketu [B]
m .. rychlost sítě [mb/s]

Třídní diagram fyzického modulu je na obrázku 4.7. Fyzická vrstva vlastní všechna rozhraní (switchporty). `Switchport` je abstraktní struktura, která je rodičem všech konkrétních switchportů. Simulátor obsahuje `RealSwitchport`, který zajišťuje připojení do skutečné sítě (toto měl na starosti kolega Bc. Tomáš Pitřinec) a `SimulatorSwitchportV2`, který obsluhuje standardní komunikaci v rámci interní virtuální sítě. Tento `SimulatorSwitchportV2` pracuje ve svém vlákně, ve kterém vykonává veškerou svoji činnost. Pokud není v bufferech switchportu žádný požadavek k vyřízení, tak vlákno spí a nezatěžuje zdroje. V tomto fyzickém modulu je kabel (`CableV2`) pouze jednoduchou datovou strukturou, která jen předává pakety z jednoho konce na druhý.



Obrázek 4.7: Třídní diagram fyzického modulu

4.4 Síťový modul

Části implementace síťového modulu jsou založeny na poznatcích z projektu psimulator1. Při analýze jsem zjistil, že implementace IP stacku (windows, linux, cisco a další) zpravidla respektují většinu ze standardů popsaných dokumenty RFC. Jak byly postupně RFC dokumenty aktualizovány, tak se chování implementací začalo lišit v detailech podle toho, zda (a jak) zapracovali změny dle RFC.

Síťový modul (části, které jsem měl za úkol) jsem implementoval dle návrhu 3.5. Mým úkolem byla implementace síťové a transportní vrstvy.

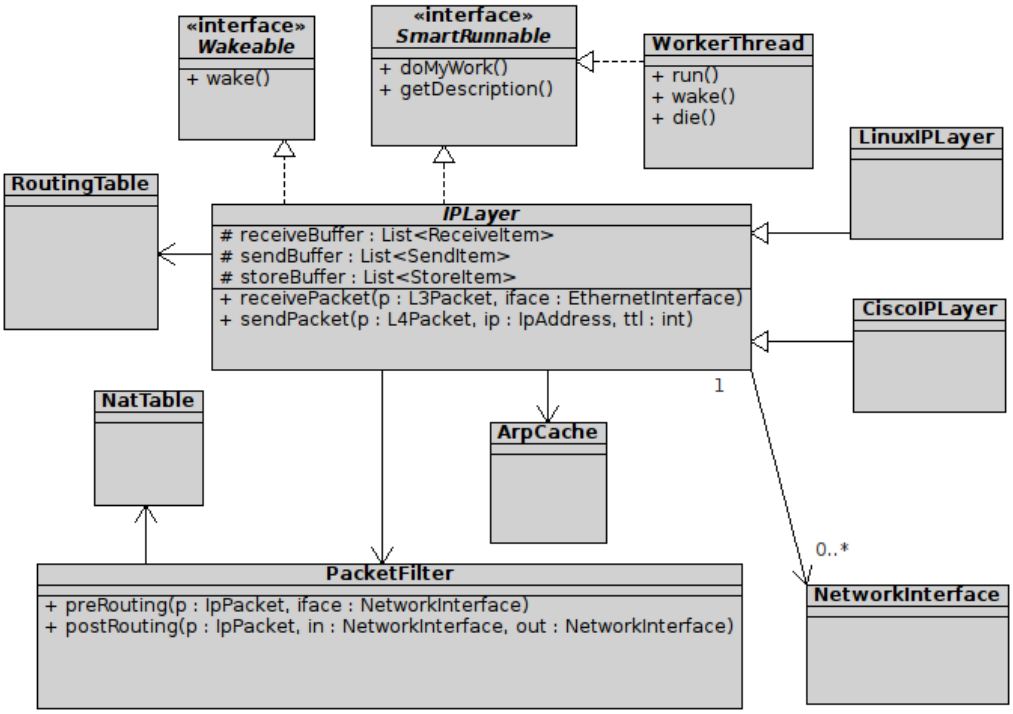
4.4.1 Linková vrstva

Linková vrstva nebyla součástí mé práce, implementoval ji kolega Bc. Tomáš Pitřinec ve své diplomové práci [30]. Z mého pohledu linková vrstva předává pakety z fyzického modulu síťové vrstvě.

4.4.2 Síťová vrstva

Síťovou vrstvu z hlediska datových struktur nejlépe popíše obrázek 4.8. Síťovou vrstvu tvoří:

- routovací tabulka (`RoutingTable`)
- paketový filter (`PacketFilter`) + překlad adres (`NatTable`)
- ARP cache (`ArpCache`)
- seznam síťových rozhraní (`NetworkInterface`)
- jádro vrstvy rozdělené do konkrétních tříd `CiscoIPLayer` a `LinuxIPLayer`



Obrázek 4.8: Třídní diagram síťové vrstvy

Síťová vrstva má k dispozici jedno pracovní vlákno (`WorkerThread`), které vyřizuje veškeré pakety (k odeslání i k přijetí). Nejdříve vysvětlím průchod paketů síťovou vrstvou „zdola“ neboli od linkové vrstvy, posléze „shora“ neboli od transportní vrstvy.

4.4.2.1 Příjem paketů od linkové vrstvy

Linková vrstva doručí příchozí paket metodou `receivePacket()`, která uloží paket do bufferu příchozích paketů (`receiveBuffer`) a probudí pracovní vlákno (pokud není již probuzené), které se postará o vyřízení paketu.

Pracovní vlákno vytáhne přijatý paket z bufferu, zjistí jakého je typu (ARP nebo IP paket) a podle něj ho zpracuje v již konkrétních metodách linuxové nebo ciscové části. Například při zpracování ARP paketu se cisco ani nesnaží poslat `ARP reply` (když je cílem dotazu), pokud nemá na odesílatele záznam v routovací tabulce. Linuxový směrovač se snaží odpověď vždy, když je cílem dotazu. Nebo při zpracování IP paketu se na linuxu kontroluje, zda cílem paketu není adresa z rozsahu 127.0.0.0 - 127.255.255.255, což by znamenalo, že adresátem je tento prvek i když nemá tuto adresu na nějakém svém rozhraní. Této funkcionalitě se říká loopback, což je virtuální síťové rozhraní, které je implementováno ve windows i na linuxu. Cisco rozhraním loopback nedisponuje. Dle RFC 3330 [19] bývá pro loopback přiřazena pouze adresa 127.0.0.1, na mém linuxu však rozhraní loopback odpovídalo na celém rozsahu 127.0.0.0/8, proto jsem to implementoval v simulátoru stejně.

Příjem IP paketu

Při příjmu IP paketu se nejdříve kontroluje, zda není cíl paketu broadcastem vůči příchozímu rozhraní. Když se tak stane, tak je paket ihned doručen transportní vrstvě.

Paket je zpracován nejdříve paketovým filtrem (metoda `prerouting()`, který provede všechna svoje pravidla[33] a předá paket NAT tabulce, která provede případný zpětný překlad cílové adresy.

Poté se zkontroluje zda cílem paketu je nějaké lokální rozhraní. V případě pozitivního nálezu je paket předán transportní vrstvě. Jinak se dekrementuje hodnota TTL. Pokud se hodnota zmenší na číslo 0, tak je poslána zpráva `Time To Live Exceeded` odesílateli. Pak se provede „zaroutování“ paketu: Pokud existuje záznam cílové sítě paketu v routovací tabulce, tak je paket předán k odeslání. V jiném případě se odešle zpráva `Destination Network Unreachable`.

4.4.2.2 Příjem paketů od transportní vrstvy

Při příjmu paketu od transportní vrstvy se:

- linux - zkontroluje zda není paket pro lokální rozhraní (pak je přijat stejným způsobem jako v 4.4.2.1), pak se provede „zaroutování“ paketu (v případě, že routovací tabulka nemá záznam pro cíl, je paket zahozen) a je předán k odeslání

[33] Žádná pravidla zatím nejsou implementována, mohou být přidána v další verzi programu.

- cisco - zkontroluje zda má záznam v routovací tabulce pro cíl paketu, pokud nemá, tak je paket zahozen

Dále probíhá zpracování paketu takto:

1. postrouting - na linuxu např. iptables zpracuje paket a předá řízení NAT tabulce, která provede případný překlad zdrojové adresy
2. zjištění linkové MAC adresy nextHopu[34] - zajišťuje ARP protokol (viz 4.4.2.3)
3. odeslání paketu

4.4.2.3 ARP protokol

Při odeslání paketu je nutné znát MAC adresu nextHopu. Pokud MAC adresa nextHopu není v době odeslání známa (nenachází se v ARP cache), tak je paket přesunut do fronty paketů čekajících na zjištění MAC adres nextHopů a zároveň je odeslána žádost - `ARP request` na linkový broadcast. Pokud nepřijde odpověď do nějaké doby[35], tak je paket vyzvednut z této fronty a je odeslána zpráva `Destination Host Unreachable`.

Bližší popis ARP protokolu je v návrhu kapitola 3.2.5.

4.4.2.4 Vlastní implementace

ARP cache jsem implementoval jako jednoduchou datovou strukturu, která disponuje metodami pro přidávání, vyzvedávání a aktualizaci záznamů. Záznamy v ARP cache jsou ukládány do hash mapy, kde klíčem je IP adresa a rozhraní, na které se daný záznam váže. Hodnotou mapy je MAC adresa a časové razítko, které používám pro zneplatňování prošlých záznamů.

Jelikož psimulator2 zatím nedisponuje funkcionalitou ARP announcements, tak jsem nastavil platnost záznamu v ARP cache na 20 vteřin. Po uplynutí této doby se bude muset odesílající znovu zeptat pomocí ARP protokolu na MAC adresu nextHopu.

Při testování simulátoru jsem objevil chybu, která způsobovala ukončení vlákna síťové vrstvy. Dlouho jsem nemohl na chybu přijít, přestože jsem danou část kódu opatřil blokem try - catch. Kvůli přechodu psimulator2 na nejnovější verzi Javy - JRE 7 jsem nejdříve podezříval Javu, nicméně jsem později zjistil, že chybu způsobuje můj kód. V metodě `handleStoreBuffer()` jsem procházel jednotlivé odložené pakety, zda k nim již neznám MAC adresu nextHopu. Při vypršení záznamu se odesílá zpráva `Destination Host Unreachable`. Aby ji bylo možné odeslat, tak je nutné znát MAC adresu nextHopu pro tuto zprávu, což někdy znamená odeslání `ARP request` a uložení této zprávy

[34] NextHop je nejbližší uzel na cestě k cíli paketu.

[35] Jednotlivé ARP záznamy mají časové razítko, které zaručuje, že jejich platnost vyprší v nastaveném čase.

do `storeBufferu`. Jelikož jsem ale procházel `storeBuffer` v cyklu, v jehož vnitřku jsem měnil data této struktury, tak java vyhazovala výjimku ConcurrentModificationException. Výjimka padala na dvou místech: v cyklu, který zpracovával `storeBuffer` a v místě přidávání do tohoto bufferu. Nechytnutí výjimky bylo způsobeno při přidávání do tohoto bufferu. Po přepsaní zpracování `storeBufferu` a přidání synchronizace na buffer tento problém zmizel.

4.4.2.5 Překlad adres

Překlad adres je z velké části založen na mé práci z projektu psimulator1. Všechny metody jsem přeložil do angličtiny kvůli rozšiřitelnosti projektu za hranice českého jazyka. Přepracoval jsem interní datové struktury, aby lépe odpovídali celkovému návrhu simulátoru. Dále jsem rozdělil ukládání dynamických a statických záznamů, protože nemají mnoho společného. U statických záznamů eviduji:

- zdrojová adresa před překladem
- zdrojová adresa po překladu

U dynamických záznamů si ukládám:

- InnerRecord před překladem
- InnerRecord po překladu

InnerRecord je nová datová struktura, která zastřešuje IP adresu, číslo portu a typ paketu čtvrté vrstvy (= typ protokolu). Zjednodušený diagram je na obrázku 4.9.

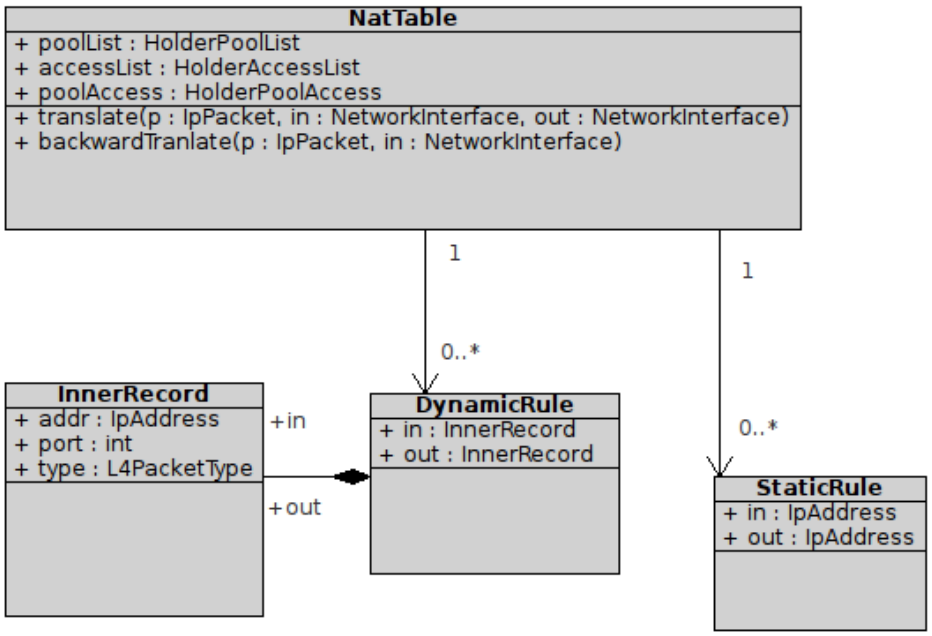
Celý překlad adres je implementován uvnitř paketového filtru v rámci metod `prerouting()` (dopředný překlad adres, „zanatování“) a `postRouting()` (zpětný překlad adres, „odnatování“). Jakmile jsou zavolány tyto metody v paketovém filtru, tak se rovněž provede i překlad adres.

Přesný popis fungování implementované NAT tabulky je v mé bakalářské práci [34].

4.4.3 Transportní vrstva

Transportní vrstva je v simulátoru natolik jednoduchá, že může pracovat ve vláknech síťové vrstvy či aplikace odesílající paket bez nutnosti práce ve vlastním vlákně. Tato vrstva je tu vložena z důvodu doručování paketů aplikacím poslouchajících na příslušných portech a zpracovávání `ICMP request` zpráv.

Přijatý ICMP paket je předán `IcmpHandleru`, který se stará o odeslání odpovědi na `ICMP request` a přeposlání ostatních zpráv aplikaci. Transportní vrstva neví, komu by měla přijatý ICMP paket předat (jaké aplikaci, na kterém



Obrázek 4.9: Zkrácený třídní diagram NAT tabulky

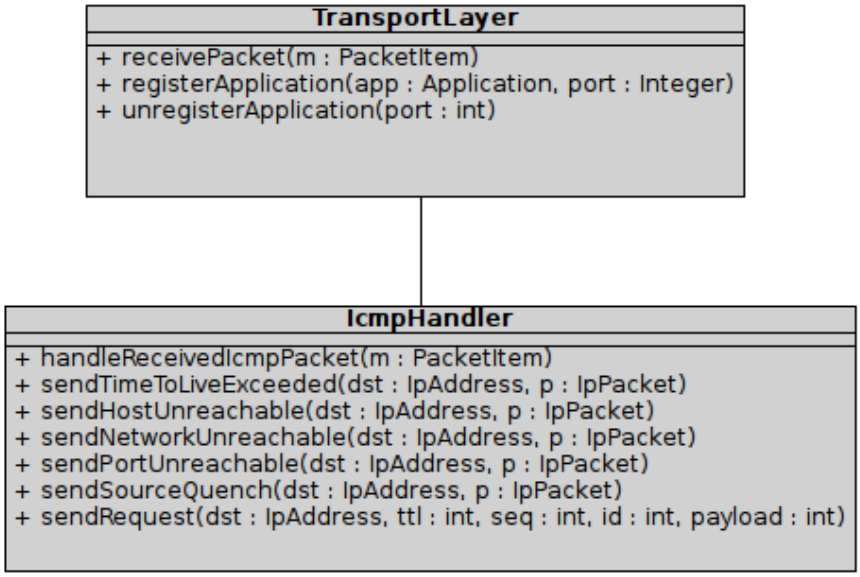
portu), protože nic o ICMP paketech neví. O to se stará `IcmpHandler`, který podle `Identifier` rozhodne, na který port (na kterou aplikaci) daný paket předat.

`Identifier` neboli identifikátor aplikace je ve skutečnosti posílán pouze u `ICMP request` a `ICMP reply`. Aby však mohla zpráva např. `Destination Host Unreachable` projít přes NAT, tak je při odeslání nového paketu (např. Host Unreachable) původní paket přibalen za nový. Myslím, že toto chování není pro psimulator2 nutné a spokojím se s tím, že všechny ICMP pakety budou mít pole `Identifier` a `sequence`, které je přes překlad adres bez problému protáhnou.

4.5 Načítání a ukládání

Architekturu načítání a ukládání implementoval kolega Bc. Martin Lukáš, který ji blíže popisuje ve své diplomové práci [16]. Z mého (uživatelského) pohledu načítání pracuje následovně: (v závorkách budou jména lidí, kteří implementovali danou část)

1. načtení (deserializace) konfiguračního XML souboru - je společné pro backend i frontend (Martin Lukáš)



Obrázek 4.10: Třídní diagram transportní vrstvy

2. zpracování načtené konfigurace a vytvoření interních struktur backendu (já - Stanislav Řehák, Tomáš Pitřinec)

Druhou část jsme implementovali oba, protože každý sám si zajišťoval ukládání/načítání svých částí v projektu.

Ukládání probíhá v opačném pořadí:

1. transformace aktuální konfigurace sítě do společných struktur (já - Stanislav Řehák, Tomáš Pitřinec)

2. uložení (serializace) společných struktur do XML souboru (Martin Lukáš)

Nespornou výhodou tohoto systému je fakt, že je velice jednoduché přidat nový atribut do ukládacího/načítacího procesu:

1. přidání atributu do společných ukládacích struktur - projekt Shared [36] do balíčku shared.Components.simulatorConfig - velice jednoduchá operace

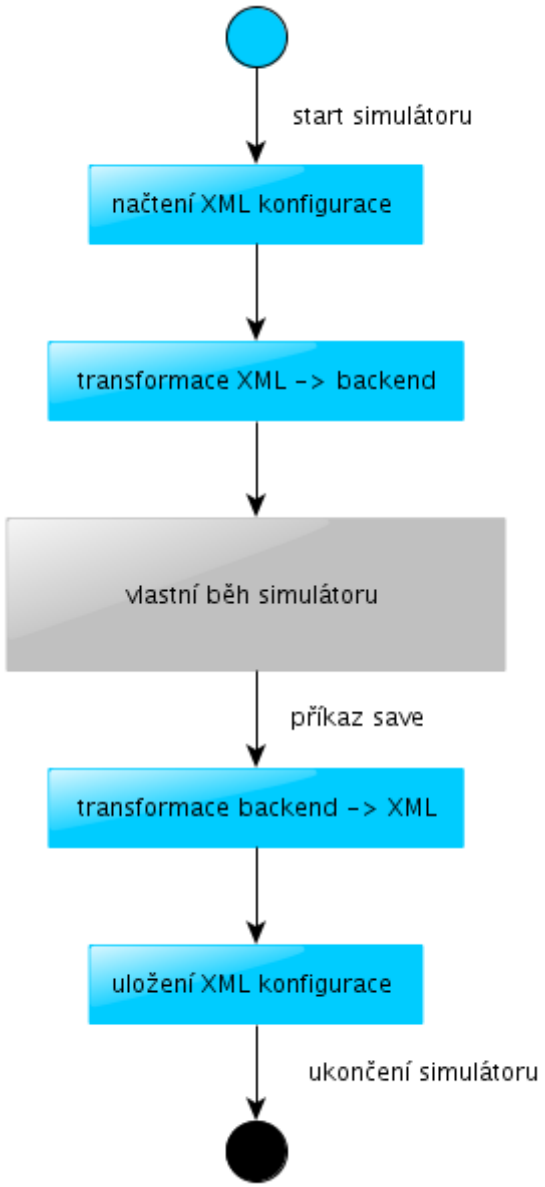
2. transformace tohoto atributu ze spuštěného backendu do projektu Shared - velice jednoduchá operace

[36] Projekt Shared je balíček, který je přidáván do backendu i frontendu v době build procesu.

3. zpětná transformace z projektu Shared do backendu - netriviální operace, která může obsahovat různé kontroly pro přidání do backendu

Nutnost třetího kroku tohoto procesu je důvodem absence nějaké větší automatizace přidávání nových atributů do ukládacího/načítacího procesu.

Celý proces ukládání a načítání ilustruje obrázek 4.11.



Obrázek 4.11: Diagram procesu načítání a ukládání

4.6 Logování

Po implementaci první verze simulátoru (psimulator1) bylo jasné, že je nutné vytvořit centrální logovací systém, který by byl schopen zaznamenávat všechny provedené akce a případně na některé akce reagovat.

4.6.1 Požadavky

- Logger musí umět vypisovat zalogované zprávy na standardní výstup backend konzole nebo do souboru.
- Logger musí logovat a vypisovat zprávy z různou úrovní důležitosti.
- Logger musí umět logovat zprávy s kategorií, která k dané zprávě náleží a dle toho ji (ne)vypisovat.
- Vypisování zpráv musí být konfigurovatelné dle kategorie a důležitosti zprávy.
- Logger musí zpracovávat nejen textové zprávy, ale i objekty např.: odesílané a zahazované pakety či vyhozené výjimky
- Logger musí umožňovat předávání zpráv jiným částem systému (pakety frontendu či zprávy pro cisco `debug` příkaz)

Podílel jsem se návrhu a implementaci částí loggeru, zde popíši některé části.

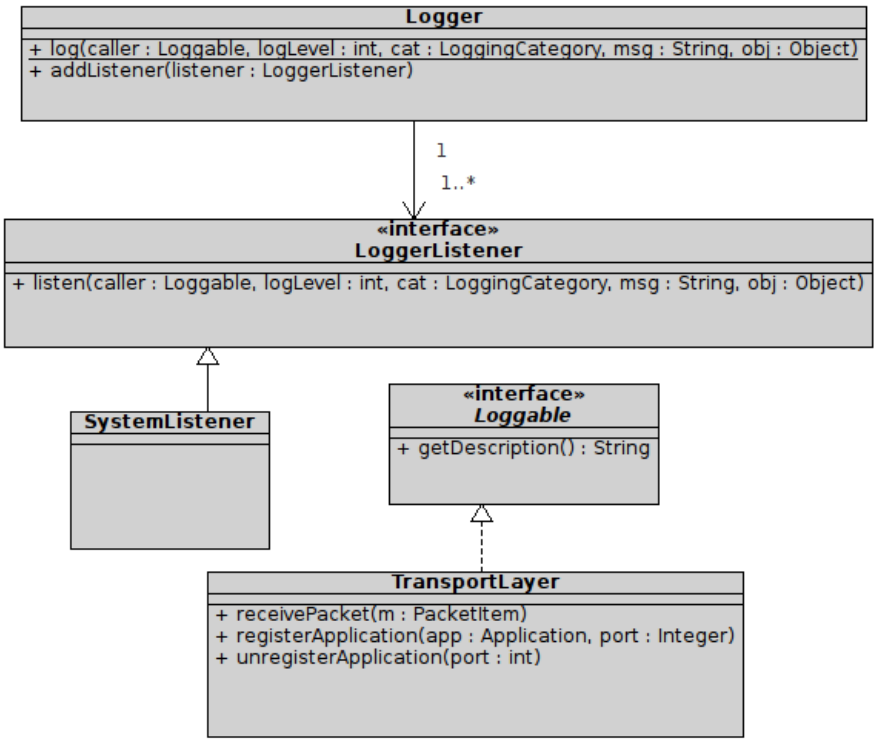
`Logger` obsahuje statické metody pro logování zpráv dle výše uvedených požadavků (důležitost a kategorie zprávy, samotná zpráva, případný objekt). Po startu simulátoru je inicializován `Logger` tak, že se vytvoří `SystemListener`, který řídí vypisování zpráv na standardní výstup. Poté se spustí `ConfigureSystemListener`, který zapíše nastavení systémového listeneru do `Logger`u. Tento krok zajistí, že je možné vypisovat jen takové zprávy, které uživatel opravdu chce. `SystemListener` má za úkol:

- logování výjimek do souboru
- předávání zalogovaných paketů[37] modulu, který odešle zprávu frontendu

Architektura loggeru umožňuje vytvoření dalších listenerů, které mohou provádět se zprávami další činnosti např.: implementace příkazu `debug` pro Cisco IOS by znamenala vytvoření listeneru, který by vypisoval všechny nastavené zprávy do konzole klienta.

Každá třída, která chce používat `Logger`, musí implementovat rozhraní `Loggable`, podle kterého systémový listener pozná, kdo danou zprávu odeslal. Na obrázku 4.12 je zobrazen návrh tříd logovací architektury.

[37] Předávájí se pakety, které se odesílají z jednoho prvku na druhý a pakety, kterou jsou z nějakého důvodu zahozeny.



Obrázek 4.12: Třídní diagram logovací architektury

V Loggeru jsem implementoval hlavně tyto části:

- předávání a transformaci odchozího paketu pro modul, který odešle danou zprávu frontendu
- zachytávání a logování výjimek

Výjimky jsou nejčastěji vyhazovány v pracovních vláknech jednotlivých vrstev (či aplikací), proto jsem implementoval mechanismus, který zajistí, že je kvůli vyhozené výjimce ztracen pouze jeden právě vyřizovaný požadavek. Celá událost je ihned zalogována do speciálního souboru[38]. Pokud je simulátor následně ukončen (a chybový výpis je z nějakého důvodu ztracen), tak je možné nalézt zalogovanou událost s časovým razítkem v souboru, jehož jméno je ve formátu `psimulator2_exceptions_yyyy-MM-dd.txt`, kde yyyy-MM-dd je aktuální datum.

[38]Přesněji řečeno je událost přidána na konec tohoto souboru, proto se případné zalogované výjimky vzájemně nepřepisují.

KAPITOLA 5

Testování

5.1 Uživatelské testování

Uživatelské testování proběhlo na 3 laboratorních cvičeních 11.4.2012 předmětu BI-PSI na FIT ČVUT. Testování bylo rozděleno na dvě části: frontend a backend. Jelikož frontend (grafické uživatelské rozhraní) není součástí mé práce, tak se tu o výsledcích této části nebudu zmiňovat a zaměřím se převážně na testování backendu.

Zúčastnění studenti dostali formuláře se scénářem testování, kde byly napsány pokyny k testování. Na tento formulář měli studenti vpisovat své návrhy ke zlepšení, odhalené chyby či neočekávané chování.

Během testování jsem prováděl pozorování a sepisoval jsem si svoje poznámky, když jsem viděl, že je nějaký problém. Testování se celkově zúčastnilo 20 lidí, jednalo se o studenty druhého ročníku bakalářského programu. Studenti prováděli testování na vlastních noteboocích pod těmito operačními systémy: různé distribuce linuxu, Windows XP a Windows 7. Znalosti studentů byly značně proměnlivé - testování se zúčastnili lidé s certifikáty Cisco Networking Academy, ale i lidé, kteří moc sítím nerozuměli. Většina studentů měla průměrné znalosti. Jednalo se tedy o ideální skupinu testerů.

5.1.1 Testovací síť

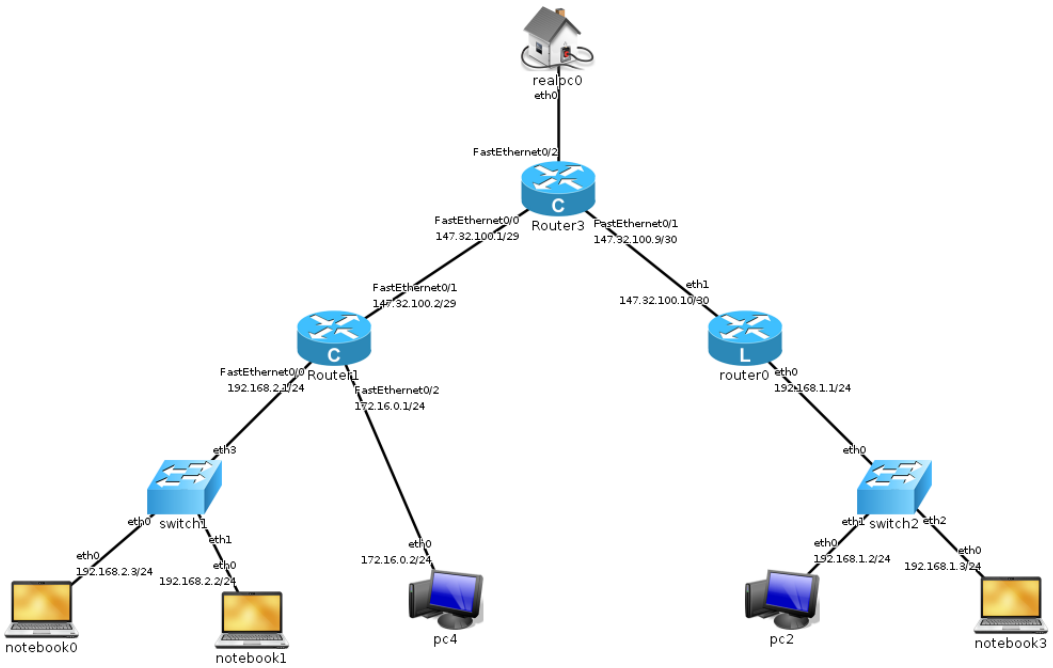
Studenti dostali pro testování speciálně vytvořenou síť[39] (`na_testovani.xml`), která obsahovala:

- 4 koncové stanice založené na linuxu
- 2 switche

[39]Testovací síť je ke stažení na: `http://code.google.com/p/psimulator/source/browse/trunk/psimulator2/src/xml/na_testovani.xml?r=1079`

- 2 cisco směrovače
- 1 směrovač založený na linuxu

Popisovaná síť je zobrazena na obrázku 5.1. Směrovače a koncové stanice měli nastaveny pouze IP adresy na rozhraních, nic jiného nastaveno nebylo.



Obrázek 5.1: Síť pro uživatelské testování

5.1.2 Úkoly

5.1.2.1 Linuxová část

1. nastavte IP adresy na pc2, router0 dle obrázku
2. zprovozněte spojení mezi pc2 a router0 (pozor na správné nastavení routovací tabulky)
3. nastavte IP adresu na router3 pro rozhraní FA0/1 a nastavte maškarádu na router0 pro pc2

5.1.2.2 Cisco část

1. nakonfigurujte IP adresy na prvcích notebook1, pc4, router1

2. nakonfigurujte statický překlad adres pro prvek pc4 na router1 (pozor na správný výběr IP adresy pro překlad - musí být ve stejné síti)

3. nakonfigurujte dynamický překlad adres pro prvek notebook1 na router1

4. správné nastavení obou překladů adres ověřte pingem z notebook1 a pc4 na router3 (pozor na správné nastavení routovací tabulky)

5.1.3 Připomínky a vyhodnocení

Většinu připomínek, které se netýkají mé práce, jsem zde zamlčel, protože o ní budou psát jiní.

1. Několika studentům nešlo vůbec spustit simulátor a vypisovalo to hlášení „psimulator2.Main was not found“.

   Tito studenti nečetli instalační příručku, a tak si nenainstalovali Java Runtime Environment ve verzi 7. Spouštění simulátoru by muselo být řešeno skriptem, který byl kontroloval verzi Javy přes samotným spuštěním simulátoru. To by ale znamenalo, že by simulátor musel obsahovat spouštěcí skripty pro všechny operační systémy. V blízké budoucnosti bude však nová verze JRE distribuována v rámci automatických aktualizací od Oraclu, takže tento problém postupně vymizí, a proto není nutné se jím dále zabývat.

2. Po rozbalení a otevření souboru `readme.txt` v operačním systému Windows v poznámkovém bloku jsou slity všechny řádky do jednoho.

   Tento problém je způsoben tím, že poznámkový blok neumí rozeznávat jiné konce řádků než výchozí (`\r\n`) na platformě Windows. Při otevření např. přes Total Commander a jeho Lister je vše v pořádku. Tato nepříjemnost je zatím neopravena z důvodu dobře fungující wiki stránky, kde je tento návod také.

3. Při postupování podle návodu ke spuštění frontendu na platformě Windows tento proces skončil s chybou - `java: command not found`.

   Chyba vznikla tak, že tento uživatel neměl JRE nainstalováno v systémové proměnné $PATH. U jiných uživatelů tento problém nenastal, a tak to bylo asi způsobeno uživatelovými zásahy do vlastního systému.

4. Frontend by mohl umožňovat přidávání výchozí brány.

   Tuto problematiku má za úkol Bc. Martin Švihlík, protože spadá do frontendu. Dle mého názoru je to zbytečné, simulátor byl navržen a implementován s myšlenkou, že frontend neumí síť nastavit, ale je to pouze nástroj pro vytvoření sítě.

5. Simulátor by mohl podporovat různé typy kabelů - přímé vs. křížené.

   Jelikož většina dnešních síťových prvků si umí kabely detekovat samo a případně si je samo přehodí. Tato problematika se neřeší při laboratorních cvičeních, pro které byl tento simulátor navržen.

6. Ve frontendu jsou špatně čitelné řádky v tabulce u označených ztracených paketů.

   Na tuto chybu si stěžoval jen jeden člověk a dle mého názoru to je čistě subjektivní záležitost. Naopak jsem zaznamenal pozitivní komentáře, že je vše přehledné.

7. Zjednodušit spuštění backendu.

   Nejdříve se musí pustit frontend pro vytvoření topologie sítě, pak se musí (ručně) spustit backend a pak je teprve možné se připojit k serveru (backendu) a zobrazovat zachycené pakety. Spuštění backendu z frontendu není implementováno, nicméně je to určitě dobrý nápad a je to jedna z věcí, která by se měla časem implementovat.

8. Na linuxu nefungoval ping na loopback (na adresu 127.0.0.1).

   Tato funkcionalita byla doplněna.

9. Někomu prý nefungoval příkaz `save` na ciscu.

   Tato nefunkčnost byla způsobena jinou chybou (pravděpodoně byla vyhozena nějaká výjimka, díky které přestala fungovat část programu).

10. Na ciscu nefungoval příkaz `interface FA0/0` v config módu.

    Tento příkaz byl po testování implementován.

11. Na ciscu nefungoval skok na jiné rozhraní při nastavování jiného rozhraní (`interface FA0/1`) v Conlon-if módu.

    Tento příkaz byl po testování implementován.

12. Na ciscu chybí příkaz `show ip interface`.

    Tento příkaz byl po testování implementován.

13. Na ciscu chybí příkaz `show ip interface brief`.

    Tento příkaz byl po testování implementován.

14. Na ciscu není implementován příkaz `?` - nápověď v Cisco IOS.

    Tento příkaz je ve fázi plánování, zatím ale nebyl implementován. Místo něho je možné použít interní příkaz `help`, který vypíše seznam podporovaných (implementovaných) příkazů.

15. Na ciscu funguje zvláštně přepínání historií při vstupu do jiného módu.

    V době testování nebyla tato funkcionalita ještě hotova, nicméně v současné době již vše funguje.

16. Dle jednoho uživatele je simulátor spustitelný pod OpenJDK (Open source náhrada JRE od Oraclu).

    Simulátor nebyl testován pod OpenJDK, a tak je dobré, že je psimulator2 kompatibilní i s open source verzí Javy.

17. Nefungoval ping na vlastní rozhraní (na nějakou svoji IP adresu)

    Tesně před testováním byly přidány do simulátoru změny[40], které nebyly řádně otestovány. Tyto změny zapříčinily pád části simulátoru - konkrétně síťové 3. vrstvy. Tato chyba byla opravena ještě při testování a byla vydána nová verze simulátoru, kterou si testeři mohli během pár minut stáhnout.

18. V případě, že byl nastaven překlad adres a zároveň uživatel spustil `ping` na nějakou vlastní adresu, byla vyhozena výjimka a IP vrstva přestala pracovat.

    Tato chyba byla způsobena výše popsanými změnami a také tím, že `ping` na vlastní IP adresu nebyl řádně otestován, a tak se na chybu přišlo až při tomto testování. Chyba byla opravena.

Po zkušenostech z tohoto testování jsem kvůli předcházení situací, kdy se vyhodí výjimka a nějaká vrstva přestane pracovat, implementoval mechanismus, který zařídí, že vyhozená (a neošetřená) výjimka v nějaké vrstvě nezpůsobí pád této vrstvy, ale pouze ztrátu právě vyřizovaného požadavku. Taková výjimka je vypsána v backendu a je navíc zalogována s časovým razítkem na konec souboru `psimulator2_exceptions.txt`.

5.2 Automatické testování

Platformu pro automatické testování konfigurace sítě implementoval kolega Bc. Martin Lukáš. Testování probíhá tím způsobem, že se systém postupně připojuje na jednotlivé prvky a posílá `ICMP request` na IP adresy definované ve scénáři. Vrácené odpovědi kontroluje podle předem připravených pravidel.

Scénář je blíže popsán v příloze C.

[40] Jednalo se o změny v IP vrstvě kvůli implementaci DHCP serveru, který zasahuje právě i do této vrstvy.

KAPITOLA 6

Návrhy na zlepšení

Projekt psimulator2 by mohl být vylepšen těmito funkcionalitami:

- Pro jednodušší spouštění celého simulátoru by mohla být přidána možnost spuštění backendu ve frontendu s aktuálně otevřenou konfigurací sítě.
- Přidání dalšího listeneru do logovacího systému, který by zajišťoval příkazy: linux - tcpdump, Cisco IOS - debug ip icmp.
- Implementace příkazů pro linux: `tcpdump` a pro cisco: `debug`, '`?`'.
- Implementace aplikace DNS, která by zajišťovala funkcionalitu DNS serveru.
- Implementace jednoduchého webové serveru, který by uměl vracet statické stránky pro HTTP dotazy.
- Implementace dalších síťových prvků např.: bridge nebo domácí router
- Implementace Spanning Tree protokolu (STP), který by zajišťoval ochranu před cykly na linkové vrstvě[41].

[41] Simulátor disponuje pouze jednoduchou kontrolou cyklů.

Závěr

Během této diplomové práce se podařilo navrhnout a implementovat síťový simulátor založený na směrovačích cisco. Ve spojení s ostatními částmi systému (implementované v rámci vývojového týmu) je simulátor schopen úspěšně odsimulovat počítačovou síť složenou ze switchů a směrovačů (linux i cisco) a konfiguraci prvků skrze textový režim příkazového řádku. Všechny proběhlé akce je možné zkontrolovat ve výpisu backendu nebo je možné simulátor napojit na a pomocí něho vizualizovat průchod paketů sítí. V této verzi simulátoru se podařilo zapracovat všechny požadavky, které vznikly na základě zkušeností z první verze např.: simulátor umožňuje napovídání příkazů, zvládá historii příkazů pro různé módy Cisco IOSu a správně reaguje na signály (Ctrl+C pro linux, Ctrl+Shift+6 pro cisco a jiné).

Simulátor byl navržen s ohledem na budoucí rozšiřitelnost, proto je možné jednoduše přidat další funkcionality např. přes platformu aplikací (viz kapitola 4.2.3).

Měsíc od vystavení programu na Google Code [31] má web simulátoru návštěvnost přibližně 10 uživatelů za den. Celkový počet návštěv je 392, z toho 231 je ze zahraničí. Díky internetové stránce simulátoru nás kontaktoval jeden z učitelů španělské univerzity La Laguna[42]. Na této univezitě je simulátor v současné době testován při výuce.

Celý projekt psimulator2 obsahuje přes 60 000 řádků kódu v Javě, z toho pouze samotný backend má přibližně 35 000 řádek.

Původní projekt byl nasazen do výuky, kde se osvědčil. Z toho důvodu se domnívám, že projekt psimulator2 (se všemi zmíněnými vylepšeními) bude přínosem pro předmět BI-PSI.

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

Literatura

[1] Boson Holdings, LLC.: *Cisco Network Simulator & Router Simulator*. [cit. 2012-03-20]. Dostupné z WWW: <http://www.boson.com/netsim-cisco-network-simulator>

[2] Castor Community: *The Castor Project*. [cit. 2012-04-26]. Dostupné z WWW: <http://www.castor.org/>

[3] Christophe Fillot: *Cisco 7200 Simulator*. [cit. 2012-03-20]. Dostupné z WWW: <http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator>

[4] Cisco Systems, Inc.: *Cisco IOS Command Modes*. [cit. 2012-04-28]. Dostupné z WWW: <http://www.cisco.com/en/US/docs/ios/12_2/configfun/configuration/guide/fcf019.html#wp1000889>

[5] Cisco Systems, Inc.: *Cisco IOS File System*. [cit. 2012-04-20]. Dostupné z WWW: <http://www.cisco.com/en/US/docs/ios/11_3/feature/guide/IFS.html>

[6] Cisco Systems, Inc.: *Cisco Packet Tracer*. [cit. 2012-03-20]. Dostupné z WWW: <http://www.ciscopros.info/web/learning/netacad/course_catalog/PacketTracer.html>

[7] Cisco Systems, Inc.: *Configuring Network Address Translation: Getting Started*. [cit. 2012-04-20]. Dostupné z WWW: <http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094e77.shtml>

[8] Cisco Systems, Inc.: *Configuring the Address Resolution Protocol (ARP)*. [cit. 2012-04-28]. Dostupné z WWW: <http://www.cisco.com/en/US/docs/app_ntwk_services/data_center_app_services/css11500series/v7.30/configuration/routing/guide/ARP.html#wp1015674>

[9] Cisco Systems, Inc.: *How NAT Works*. [cit. 2012-04-20]. Dostupné z WWW: <http://www.cisco.com/en/US/tech/tk648/tk361/technologies_tech_note09186a0080094831.shtml>

[10] Cisco Systems, Inc.: *Interface Commands (show ip interface)*. [cit. 2012-04-28]. Dostupné z WWW: <http://www.cisco.com/en/US/docs/ios/12_0/interface/command/reference/irshowip.html#wp1028374>

[11] Cisco Systems, Inc.: *Using the traceroute Command on Operating Systems*. [cit. 2012-04-30]. Dostupné z WWW: <http://www.cisco.com/en/US/tech/tk364/technologies_tech_note09186a00801ae32a.shtml#addno>

[12] Department of Computer Science and Information Systems, Birkbeck, University of London: *Address Resolution Protocol*. [cit. 2012-04-21]. Dostupné z WWW: <http://penguin.dcs.bbk.ac.uk/academic/networks/network-layer/arp/index.php>

[13] GNS3 Community: *GNS3*. [cit. 2012-04-20]. Dostupné z WWW: <http://www.gns3.net/>

[14] Greg Anuzelli: *Dynagen - The network configuration generator for Dynamips*. [cit. 2012-04-20]. Dostupné z WWW: <http://dynagen.org/>

[15] Internet Assigned Numbers Authority (IANA): *Internet Control Message Protocol (ICMP) Parameters*. [cit. 2012-04-20]. Dostupné z WWW: <http://www.iana.org/assignments/icmp-parameters/icmp-parameters.xml>

[16] Lukáš, M.: *Podpůrné komponenty simulátoru počítačové sítě: Diplomová práce*. České vysoké učení technické v Praze, 2012.

[17] Michael C. Toren: *tcptraceroute*. [cit. 2012-04-29]. Dostupné z WWW: <http://michael.toren.net/code/tcptraceroute/>

[18] Michek, J.: *Emulátor počítačové sítě: Diplomová práce*. České vysoké učení technické v Praze, 2008.

[19] Network Working Group: *RFC 3330: Special-Use IPv4 Addresses*. [cit. 2012-04-30]. Dostupné z WWW: <http://tools.ietf.org/html/rfc3330#page-2>

[20] Network Working Group: *RFC 5227: IPv4 Address Conflict Detection*. [cit. 2012-04-23]. Dostupné z WWW: <http://tools.ietf.org/html/rfc5227#page-15>

[21] Network Working Group: *RFC 768: User Datagram Protocol*. [cit. 2012-04-24]. Dostupné z WWW: <http://tools.ietf.org/html/rfc768>

[22] Network Working Group: *RFC 791: Internet Protocol*. [cit. 2012-04-24]. Dostupné z WWW: <http://tools.ietf.org/html/rfc791#page-11>

[23] Network Working Group: *RFC 792: Internet Control Message Protocol: Echo or Echo Reply Message*. [cit. 2012-04-24]. Dostupné z WWW: <http://tools.ietf.org/html/rfc792#page-14>

[24] Network Working Group: *RFC 792: Internet Control Message Protocol: Source Quench Message*. [cit. 2012-05-2]. Dostupné z WWW: <http://tools.ietf.org/html/rfc792#page-10>

[25] Network Working Group: *RFC 793: Transmission Control Protocol*. [cit. 2012-04-24]. Dostupné z WWW: <http://tools.ietf.org/html/rfc793#page-15>

[26] Network Working Group: *RFC 826: An Ethernet Address Resolution Protocol*. [cit. 2012-04-21]. Dostupné z WWW: <http://tools.ietf.org/html/rfc826>

[27] NS Community: *ns-3*. [cit. 2012-03-09]. Dostupné z WWW: <http://www.nsnam.org/>

[28] OMNeT++ Community: *OMNeT++ Community Site*. [cit. 2012-04-01]. Dostupné z WWW: <http://www.omnetpp.org/>

[29] Oracle Corporation: *Oracle VM VirtualBox*. [cit. 2012-04-20]. Dostupné z WWW: <https://www.virtualbox.org/>

[30] Pitřinec, T.: *Síťový simulátor pro výukové účely na bázi prvků OS Linux: Diplomová práce*. České vysoké učení technické v Praze, 2012.

[31] Psimulator developers: *psimulator - Simple graphical linux and cisco network simulator*. [cit. 2012-04-30]. Dostupné z WWW: <http://code.google.com/p/psimulator/>

[32] QEMU Community: *QEMU - open source processor emulator*. [cit. 2012-04-20]. Dostupné z WWW: <http://wiki.qemu.org/Main_Page>

[33] ZOHO Corp.: *WebNMS Simulation Toolkit 7*. [cit. 2012-03-20]. Dostupné z WWW: <http://www.webnms.com/simulator/index.html>

[34] Řehák, S.: *Simulátor virtuální počítačové sítě Cisco: Bakalářská práce*. České vysoké učení technické v Praze, 2010.

PŘÍLOHA A

Seznam použitých zkratek

ARP Address Resolution Protocol

BI-PSI Počítačové sítě

CCNA Cisco Certified Network Associate

CNA Cisco Networking Academy

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DRAM Dynamic Random Access Memory

FTP File Transfer Protocol

GUI Graphical user interface

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDE Integrated Development Environment

IP Internet Protocol

IOS Internetwork Operating System

JRE Java Runtime Environment

MAC Media Access Control

NAT Network Address Translation

P2P Peer to peer

RFC Request for Comments

STP Spanning Tree Protocol

SSH Secure Shell

TCP Transmission Control Protocol

TFTP Trivial File Transfer Protocol

TTL Time To Live

UDP User Datagram Protocol

UML Unified modeling language

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

Příručka pro napojení na reálnou síť je v diplomové práci Bc. Tomáše Pitřince [30].

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.

PŘÍLOHA C

Scénář automatického testování

pc1
- 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`)

pc0
- ping -c1 8.8.8.8 (očekává „connect: Network is unreachable“)

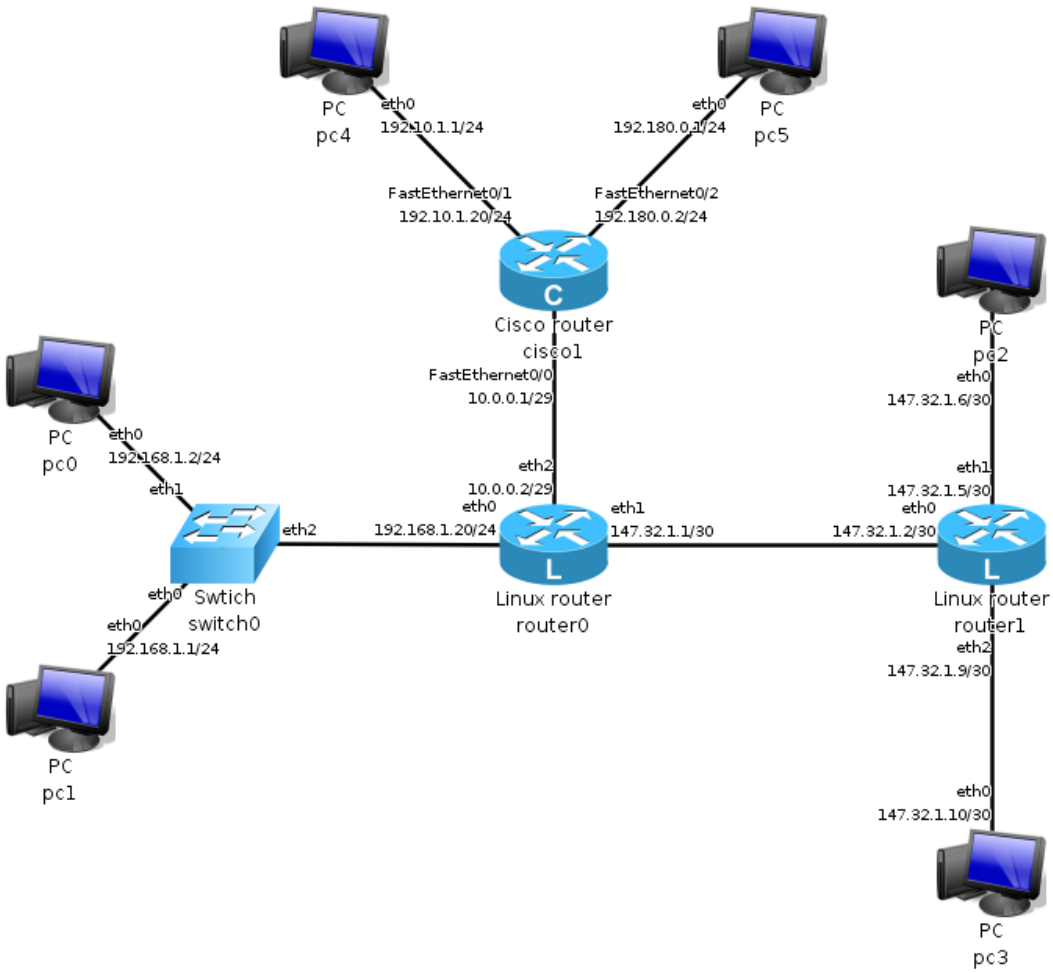
pc4
- ping -c1 192.168.1.1 (očekává „64 bytes from 192.168.1.1: icmp_req=1 ttl=62“)

pc5
- ping -c1 192.168.1.1 (očekává „64 bytes from 192.168.1.1: icmp_req=1 ttl=62“)

Testovací síť je na obrázku C.1.

C.1 Poznámky k jednotlivým prvům

pc1: má výchozí bránu na router0
pc0: nemá výchozí bránu
router0: nastavena maškaráda na odchozí rozhraní eth1, výchozí bránu na router1



Obrázek C.1: Síť pro automatické testování

router1: není nastavena 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 výchozí bránu (pro testování, kdy se nemá vrátit nic)
pc4: má výchozí bránu na cisco1
pc5: má výchozí bránu na cisco1
cisco1: má výchozí bránu na router0, statický překlad adres pc4 na 10.0.0.6, dynamický překlad adres pc5 na 10.0.0.5

PŘÍLOHA D

Obsah přiloženého CD

Přiložené CD obsahuje následující adresáře a soubory:

- `readme.txt` ..... stručný popis obsahu CD
- `bin` ..... adresář se spustitelnou formou implementace
- `src` ..... zdrojové kódy implementace
- `text` ..... text práce
  - `src` ..... zdrojová forma práce ve formátu LaTeX
  - `thesis.pdf` ..... text práce ve formátu PDF