Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Rozšíření síťového simulátoru o připojení do reálné sítě

Václav Mach

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

15. května 2014

Poděkování

Děkuji panu Ing. Pavlu Kubalíkovi, Ph.D. za zajímavý námět bakalářské práce a všestrannou pomoc při její realizaci. Dále děkuji za pomoc všem, kteří aplikaci testovali, protože výrazně pomohli zvýšit její kvalitu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) 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, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 15. května 2014 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Mach, Václav. *Rozšíření síťového simulátoru o připojení do reálné sítě*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

Abstrakt

Tato práce se zabývá analýzou, návrhem a implementací rozšíření existujícího síťového simulátoru Psimulator2 řesící připojení do reálné síťe. Simulátor má sloužit především pro výukové účely. Aplikace umožňuje uživateli vytvořit virtuální počítačovou síť složenou z různých prvků, které je možné konfigurovat pomocí příkazů vestavěného telnet rozhraní. Rozšíření má umožnit komunikaci mezi vytvořenou virtuální sítí a reálnou sítí. Protože je simulátor plně portabilní, je požadováno, aby bylo rozšířenou aplikaci možné použít na operačních systémech Microsoft Windows a Linux.

Klíčová slova síťový simulátor, reálná síť, virtuální síť, Java, Linux, Microsoft Windows, propojení sítí

Abstract

This bachelor thesis looks into analysis, design and implementation of extension of existing network simulator Psimulator2. The extension solves connection with real network. Simulator is intended primarily for educational purposes. The application allows user to create virtual computer network composed from various elements, which can be configured using commands of built-in telnet interface. The extension should allow communication between

created virtual network and real network. The simulator is fully portable, so it is required that the extended application is executable in Microsoft Windows and Linux.

Keywords network simulator, real network, virtual network, Java, Linux, Microsoft Windows, network interconnection

Obsah

Seznam obrázků

Úvod

Pro zlepšení výuky předmětu počítačové sítě (BI-PSI) na Fakultě informačních technologií ČVUT byl již dříve vytvořen síťový simulátor Psimulator. Ten byl následně rozšířen o další funkcionalitu, a tak vznikla nová verze simulátoru Psimulator2. Zásadním nedostatkem první verze simulátoru bylo neexistující grafické uživatelské rozhraní, které by umožňovalo editaci síťové topologie. Tento nedostatek byl spolu s dalšími ve druhé verzi odstraněn.

Ačkoliv Psimulator2 nabízí velké množství možností konfigurace různých síťových topologií a technologií, všechny dostupné možnosti jsou pouze zlomkem toho, co je možné vytvořit v reálném prostředí. V důsledku toho existuje stále mnoho možností, jak simulátor rozšiřovat. Aplikace je jako celek navržena velmi kvalitně a její struktura je dostatečně intuitivní pro další možnosti rozšíření. Podle mého názoru by zejména uživatelé, kteří se síťovými technologiemi začínají, ocenili rozšíření méně pokročilých technologií – například dynamických směrovacích protokolů nebo objevovacích protokolů linkové vrstvy (LLDP[1]).

Vlastní simulátor je psán v jazyce Java, v důsledku toho bude rozšíření realizováno ve stejném jazyce. Minimální požadovaná verze jazyka je Java Standard Edition 7 (1.7). (V době psaní práce dostupná Java Standard Edition 8.) Jelikož jedním z hlavních požadavků celého projektu je portabilita, byl pro první verzi simulátoru zvolen jazyk Java, který díky své architektuře používající JVM[2] tento požadavek splňuje. V následujících verzích bylo použití programovacího jazyka včetně verze zachováno.

Cílem této bakalářské práce je rozšířit simulátor o možnost propojení s reálnou sítí za použití existujícího ethernetového rozhraní, které má být realizováno na úrovni linkové vrstvy. Tím je jednoznačně stanoveno, že nebude

[1] LLDP – Link Layer Discovery Protocol. Jedná se o protokol linkové vrstvy, který umožňuje zařízením oznamovat svou identitu, schopnosti a své sousedy v lokální síti, nezávisle na výrobci.

[2] JVM – Java virtual machine. JVM je virtuální stroj, který umožňuje spuštění Java bajtkódu, jenž je výsledkem kompilace zdrojových kódů jazyka.

možné použít standardní systémové API[3]. Standardní síťové systémové API umožňuje použití síťové vrstvy a vyšších vrstev, které následují za linkovou vrstvou v ISO[4] OSI modelu[5]. V důsledku toho bude nutné využít nestandardní prostředky, které dovolují komunikovat už na úrovni linkové vrstvy.

Rozšíření by mělo být možné použít na operačních systémech Microsoft Windows a Linux. Díky podobnosti architektury jádra operačního systému Linux a operačního systému Mac OS[6] by měla v obou systémech nová funkcionalita pracovat korektně. V závislosti na možnostech, které architektura konkrétního operačního systému poskytuje, bude možné využít konkrétní specifické vlastnosti. Zároveň je třeba dbát na dostatečně obecné řešení, protože specifické řešení by nemuselo být architekturou jiných operačních systému podporováno.

Na projektu spolupracuji s Michalem Horáčkem. V rámci své bakalářské práce rozšiřuje simulátor o další funkcionalitu v simulovaných zařízeních představujících Linux. Další důležitou součástí práce je testování. Testovat budeme jak v průběhu vývoje, tak i po jeho skončení pomocí zkušebních uživatelů (studentů fakulty), které přislíbil vedoucí práce. Takové testování by mělo pomoci odhalit nejzávažnější chyby a zároveň by měl být výsledkem pravděpodobný pohled cílového uživatele.

Simulátor obsahuje vestavěnou nápovědu, kterou bude třeba doplnit o nové informace týkající se nové funkcionality. Součástí výsledného produktu by měly být i případné instalační instrukce (v případě složitějšího postupu), dále dokumentace ve formě PDF[7], pro případ kdy vestavěná nápověda nevyhovuje a nakonec příklady použití demonstrující použití konkrétních technologií. Vzhledem k jazykové lokalizaci celé aplikace by bylo vhodné, aby byly externí součásti dostupné v odpovídajících jazykových verzích.

[3]API – Application programming interface. API je rozhraní, které definuje jak spolu komunikují softwarové komponenty.

[4]ISO – International Organization for Standardization. ISO je světovou federací národních normalizačních organizací.

[5]OSI model – Open Systems Interconnection model. Jde o koceptuální model, který charakterizuje a standardizuje interní funkce komunikačního systému tím, že ho rozděluje do abstraktních vrstev. V celé práci jsou používany názvy vrstev z tohoto modelu, přestože to není explicitně uvedeno, protože v praxi se toto názvosloví používá stejným způsobem.

[6]Mac OS je operační systém používaný na zařízeních vyráběných společností Apple.

[7]PDF – Portable Document Format. Jedná se o souborový formát pro ukládání dokumentů nezávisle na softwaru, hadrwaru a operačních systémech.

Kapitola 1

Popis problému, vymezení cílů

1.1 Architektura simulátoru

První verze simulátoru měla komponentu pouze jednu – vlastní simulátor. Absence grafického uživatelského rozhraní pravděpodobně odradila velké množství potenciálních uživatelů, přestože vlastní simulátor již v té době disponoval základní funkcionalitou. Aplikace bez grafického rozhraní jsou v dnešní době stále velmi hojně využívané, ale dle mých zkušeností se téměř výhradně používají na všestranné řešení úkolů, které je nutné automatizovat. Naproti tomu aplikace s grafickým rozhraním se používají většinou na jednorázové nebo málo opakované úkony a taktéž jsou o mnoho přívětivější pro běžné uživatele. V tomto konkrétním případě mělo chybějící grafické prostředí významný dopad na míru užívání aplikace, což dokazuje druhá verze, kam bylo grafické prostředí začleněno.

Druhá verze simulátoru je tvořena dvěma hlavními komponentami – backendem a frontendem. Frontend je grafické prostředí, které umožnuje uživateli vytvářet nebo měnit síťovou topologii. Konkrétní seznam navrhovaných změn pro druhou verzi je možné dohledat v pracích autorů této verze [14] [18]. Podle mého názoru bylo největším přínosem právě grafické uživatelské rozhraní, které pravděpodobně přimělo mnoho uživatelů k vyzkoušení aplikace.

Grafické rozhraní obsahuje prvky pro manipulaci s vlastní síťovou topologií – přepínače, směrovače, kabely a koncové stanice. Uživatel má možnost si vybraná zařízení rozmístit dle libosti na pracovní ploše. U každého simulovaného zařízení je možné zobrazit vlasnosti, kde jsou dostupné informace o názvech a počtu síťových rozhraní. Dále je uveden stav linky na fyzické vrstvě, IP[8] adresa včetně masky sítě a adresa linkové vrstvy, kterou je možné změnit z původní hodnoty. Aby nebylo nutné vždy podrobně zkoumat vlastnosti každého zařízení v topologii, je u každého zařízení uveden popisek s informacemi.

[8] IP – Internet Protocol. IP je základním protokolem pracujícím na síťové vrstvě používaným v počítačových sítích, dvěma hlavními verzemi jsou IP verze 4 a IP verze 6.

Díky propojení s backendem poskytuje frontend mnohé názorné funkce. Podle mě jsou dvěma nejnázornějšími funkcemi možnost vizualizace paketů či rámců ve vlastní topologii v reálném čase a simultální zobrozování hlaviček stejných dat. Tyto dvě funkce jsou podle mého názoru pro začátečníky v oblasti sítí neocenitelné, protože velmi dobře ilustrují vše, co se děje. Další podstatnou funkcí je možnost spuštění konzolového připojení libovolného zařízení. Přestože je možné určité prvky nakonfigurovat ve vlastním grafickém prostředí, uživatel by měl usilovat o to, aby byl schopen toho docílit právě pomocí nabízeného CLI[9] rozhraní. Ačkoliv jsou konkétní CLI rozhraní oproti reálným značně méně obsáhlá, simulují reálná zařízení velmi věrně.

Backend je vlastní simulátor sítě, s grafickým prostředím je propojen pouze přes soubor ve formátu XML[10] , který obsahuje vlastní informace o topologii a konfiguraci zařízení. Protože se jedná o CLI součást celé aplikace, je její výstup směrován na výstup emulátoru terminálu, ve kterém je backend spuštěn. Podle mého názoru by pro uživatele bylo efektivnější, kdyby měl možnost si zobrazit výstup backendu v rámci grafického rozhraní. Tato modifikace by nijak neměla poškodit možnosti backendu, pokud by měl být spuštěn odděleně. Myslím, že zejména při hledání chyb v navržené topologii by uživatel ocenil, kdyby nemusel mít spuštěno více oken a mohl vše v reálném čase sledovat v grafickém prostředí simulátoru.

Obě části simulátoru je možné propojit pomocí síťového spojení na hostitelském počítači. Ve druhé verzi simulátoru je nutné spustit obě části odděleně a následně je propojit, aby bylo možné využít veškerou dostupnou funkcionalitu. Přestože je grafické uživatelské rozhraní navrženo velmi intuitivně a odpovídá současným nárokům, podle mého názoru by mohlo být propojení obou komponent realizováno lépe. Navržená architektura umožňuje jejich používání nezávisle na sobě, což může být v některých případech užití výhodou. V případě, že chce uživatel použít obě komponenty dohromady, musí s nimi pracovat odděleně a následně je spojit, a to by podle mého názoru mohlo být realizováno lépe. Cílem by podle mě mělo být zachování možnosti odděleného používaní a zároveň možnost spuštění obou komponent jako celku.

1.2 Propojení s reálnou sítí

Řešení propojení s reálnou sítí je možné rozdělit na část operačního systému a samotnou aplikaci. Na jejich rozhraní bude záviset možný dosažený výsledek. Protože má být vylepšení realizováno v Javě, bude i její rozhraní hrát roli ve

[9] CLI – Command-line interface. Někdy též nazýváno příkazový řádek, rozhraní příkazového řádku, konzolové rozhraní nebo znakové uživatelské rozhraní. Uživatel s programy komunikuje zapisováním příkazů do poskytnutého rozhraní. Díky své architektuře dovoluje bezproblémovou automatizaci úloh.

[10] XML – Extensible Markup Language. XML je značkovací jazyk, který definuje množinu oprávnění pro strukturu souborů, která je čitelná jak pro lidi, tak pro počítače.

výsledném řešení. Na první pohled nemusí být zřejmá další podstatná součást, kterou je architektura operačního systému definující například:

- Jak je možné ovlivnit směrování a jaká má v daném operačním systému specifika.
- Jak systém pracuje se síťovými rozhraními.
- Jaké typy síťových rozhraní je možné vytvořit.
- Možné operace se síťovými rozhraními.
- Jakým způsobem probíhá směrování.

Operační systém poskytuje rozhraní, které dává programátorovi různé možnosti. Na základě toho, jak benevolentně je konkretní rozhraní vytvořeno, tak dobré či špatné možnosti programátor má. Ze strany aplikace je možné právě toto rozhraní využít. Je požadováno, aby propojení s reálnou sítí bylo možné realizovat na operačních systémech používajících jádro Linux a operačních systémech Microsoft Windows. Zástupci obou systémů nabízejí standardní rozhraní pro práci se sítí, bohužel toto rozhraní umožňuje komunikaci až na síťové vrstvě. Aby bylo možné komunikovat už na linkové vrstvě, je třeba využít speciální rozhraní – v tomto případě pravděpodobně knihovnu.

Další typ rozhraní, které operační systém poskytuje, je uživatelské rozhraní. Přestože je možné takové rozhraní používat přímo ze zdrojových kódů aplikace a modifikovat tak chování systému, pro většinu uživatelů bude pravděpodobně pohodlnější používat takové rozhraní přímo. Podle mě má tento přístup minimálně tu výhodu, že uživatel se naučí pracovat se samotným rozhraním operačního systému, které může v budoucnu použít. Pokud by uměl použít pouze rozhraní aplikace, které bude manipulovat s rozhraním operačního systému, a neměl by aplikaci k dispozici, pravděpodobně by nevěděl, jak s rozhraním operačního systému manipulovat, a tak by například nebyl schopen modifikovat směrovací tabulku. Proto je podle mého názoru vhodné nechat systémové rozhraní oddělené od aplikace.

Architektura operačního systému může být stejně tak omezující či benevolentní jako API operačního systému. Dostupné možnosti závisí na použitém operačním systému. Pokud bychom zde narazili na zásadní problémy při snaze dosáhnout cíle, řešení by pravděpodobně bylo velmi složité. V závislosti na architektuře daného operačního systému bychom mohli být nuceni například upravit či vytvořit některé moduly operačního systému, a to by mohlo být jistě velice složité. Z toho důvodu by podle mého názoru bylo lepší se pokusit vyhnout řešením, které k takovým problémům vedou, jelikož by jejich řešení mohlo být velmi časově náročné.

1.3 Vymezení spolupráce

Jak jsem již napsal v úvodu práce, na projektu spolupracuji s Michalem Horáčkem. Michal pracuje na rozšíření zařízení simulujících operační systém s jádrem Linux. Více podrobností lze najít v samotné práci [7]. Jedná se konkrétně o tato vylepšení:

- Možnost konfigurace síťových rozhraní pomocí souborů.
- Jednoduchý DNS[11] protokol, včetně konfiguračních souborů. Návrh je založen na existujícím DNS serveru bind9.
- DHCP[12] protokol včetně konfiguračních souborů. Návrh je založen na existujícím DHCP serveru ISC DHCP.

Společně jsme zvažovali různé možnosti verzování zdrojových kódů, tak aby to pro nás bylo nenáročné a efektivní. Zároveň bylo nutné zvážit předchozí vývoj celého projektu a možnou distribuci uživatelům. Rozhodli jsme se, že v průběhu vývoje nové verze budeme sdílet verzovaný zdrojový kód pouze mezi sebou a výsledný produkt zpřístupníme spolu s dřívějšími verzemi v repozitáři na Internetu.

1.4 Zkoumání simulátoru

Při zkoumání simulátoru jsem narazil na několik problémů. Jedná se o různorodé problémy, které se mohou projevit nebo způsobit nedefinované chování programu pouze ve specifických případech, přesto by podle mě bylo vhodné takové problémy opravit. Jejich závažnost jistě není ve všech případech stejná, přesto podle mého názoru mohou být zásadní pro uživatele, který na ně narazí. Jedná se konkrétně o tyto problémy:

- Vizualizace rámců protokolu ARP[13] jsou na přepínači zpracovány sekvenčně, ačkoliv by měly být zpracovány paralelně.
- Možnost připojení na CLI rozhraní přímo v simulátoru v novém okně občas způsobuje nedefinovaný stav – například zamrznutí kurzoru nebo nemožnost potvrzení příkazu klávesou enter.

[11] DNS – Domain Name System. DNS je hierarchický distribuovaný systém pro počítače, služby nebo jiné zdroje dostupné v sítí, který umožňuje překlad doménových jmen na adresy protokolu IP.

[12] DHCP – Dynamic Host Configuration Protocol. DHCP je standardizovaný protokol pro dynamickou distribuci síťových konfiguračních parametrů (například IP adresa, výchozí brána nebo adresy DNS serverů) klientům.

[13] ARP – Address Resolution Protocol. ARP je protokol pro dynamické odvozování adres linkové vrstvy z adres síťové vrstvy.

- Při spuštění frontendu se v některých případech zobrazí pouze část loga aplikace.
- Uživatelem ukládané soubory mají formu XML dokumentů, ale není vynucena jejich přípona. Stejně tak není vynucena přípona u načítaných souborů.

Ačkoliv první problém nemůže nijak narušit integritu aplikace, předává uživateli zavádějící informace. Vzhledem k účelu aplikace je podle mého názoru nepřípustné, aby k něčemu takovému mohlo dojít. Tento problém se nemusí projevit v každé vytvořené topologii, pokud se však projeví, nemá uživatel žádnou možnost, jak ho odstranit.

Druhý problém může být pro užitavatele také velmi nepříjemný. V případě, že uživatel nebude chtít nebo nebude mít možnost použít externího klienta pro telnet, bude pravděpodobně muset zkusit spustit okno konzole konkretního zařízení několikrát, aby se mu podařilo dostat korektně fungující rozhraní. V případě, že bude mít uživatel k dispozici externího telnet klienta, může ho použít, ale aplikace by na něco takového neměla spoléhat.

Zbylé dva problémy by ve většině případů pravděpodobně neměly uživateli způsobit zásadní problémy, ale podle mého názoru zhoršují celkový dojem uživatelů z aplikace. Logo aplikace by mělo být podle mého názoru zobrazeno za každých okolností bez jakýchkoliv problémů. Java nabízí možnosti, které dovolují programátorovi jednoduchým způsobem vytvořit aplikaci tak, aby působila profesionálním dojmem, a proto si myslím, že odstranit tyto problémy by nemělo být nijak obtížné.

V této kapitole jsem popsal s jakými prostředky by měla analýza a vlastní řešení pracovat. Bude se pravděpodobně jednat především o analýzu možností operačních systému a existujících řešení. Na základě této prvotní analýzy jsem stanovil cíle, které bych chtěl v rámci bakalářské práce splnit. Cíle, které jsem si stanovil jsou:

- Vytvořit lepší návrh propojení obou komponent, tak aby se v případě potřeby aplikace jevila jako jedna komponenta.
- Opravit nalezené chyby.
- Vytvořit funkční řešení propojení s reálnou sítí, tak aby bylo použitelné v požadovaných operačních systémech a podle specifikace v zadání bakalářské práce.

Kapitola 2

Existující řešení

V této kapitole bych mohl popsat obecná existující řešení simulátorů sítě, ale vzhledem k užší specifikaci bakalářské práce se budu věnovat pouze takovým, které splňují požadavek na možnost propojení virtuální sítě s reálnou. Na základě doporučení od vedoucího jsem se zabýval i generátory síťové komunikace, ale v této kapitole budu popisovat jen simulátory sítě.

2.1 GNS3

GNS3[14] je open source (volně dostupný zdrojový kód) software, který je schopen simulovat komplexní sítě a přitom se snaží co nejvíce přiblížit fungování reálných sítí. Poskytuje intuitivní grafické uživatelské prostředí pro tvorbu a konfiguraci sítí, zároveň je velmi dobře portabilní díky implementaci v jazyce Python. Podporovanými operačními systémy jsou Microsoft Windows, Linux a Mac OS [6]. Hlavními komponentami celé aplikace jsou emulátory Dynamips, VirtualBox a Qemu.

Jednou z hlavních komponent celé aplikace je Dynamips. Jedná se o emulátor platformy Cisco IOS[15] , z toho vyplývá, že uživatel, který chce Dynamips využít by měl vlastnit zakoupený obraz IOS, aby neporušoval licenční podmínky společnosti Cisco Systems. Po sehnání takového obrazu jej stačí jednoduše nahrát do aplikace, zvolit konkrétní hardwarovou platformu a poté už lze využít veškerou funkcionlitu, kterou daná verze IOS nabízí. Podle mého názoru neexistuje žádný lepší způsob, jak simulovat libovolnou funkcionalitu dané platformy. Proto si myslím, že v tomto ohledu by měly být možnosti GNS3 nejlepší možné, jakých se dá dosáhnout.

Díky dalším klíčovým komponentám umožňuje aplikace virtualizovat nebo používat reálně nasazené virtuální počítače. GNS3 podporuje virtualizaci tech-

[14]GNS3 – Graphical Network Simulator 3.

[15]Cisco IOS – Cisco Internetwork Operating System. Cisco IOS je operační systém velké většiny přepínačů a směrovačů společnosti Cisco Systems.

nologií VirtualBox, KVM[16] a Qemu. Díky těmto možnostem je možné používat operační systém JunOS společnosti Juniper, který je operačním systémem síťových zařízení zmíněné společnosti. Díky technologii Qemu je možné do topologie začlenit Cisco ASA[17] , PIX[18] a IPS[19] .

Díky této funkcionalitě umožňuje uživateli vytvořit síťovou topologii, do které je možné zakomponovat reálné virtuální stroje, komplexní zařízení a téměř cokoliv dalšího, co je možné použít v reálné síťové topologii. V porovnání s ostatními síťovými simulátory tak GNS3 nabízí nepřeberné možnosti.

Takové možnosti dovolují jednoduše velmi věrně simulovat reálnou síť a zkoušet různorodé situace nebo testovat různá vylepšení, aniž by byla reálná síť nějak ovlivněna. Pro reálnou síťovou infrastrukturu to znamená možnosti testovaní možných rozšíření sítě, bezpečnosti sítě nebo vytížení sítě používáním různých služeb, a to vše bez nutnosti jakýchkoliv změn či nutného hadrwaru.

GNS3 také nabízí možnost propojení simulované síťe s reálnou. Při použití této funkcionality se jeví hostitelský počítač jako transparentní. V samotné topologii je třeba použít cloud (obecný symbol repzentující libovolnou síť). Cloud je třeba správně nakonfigurovat. Po zvolení vlastností cloudu je třeba v záložce NIO (network input output) ethernet nutné vybrat rozhraní hostitelského počítače, které bude sloužit pro výstup do reálné sítě. Následně už stačí zapojit cloud do zbytku topologie.

2.2 Psimulator2

Architekturu simulátoru jsem popsal v minulé kapitole. V této podkapitole se budu detailněji věnovat backendu a realizaci propojení s reálnou sítí. Ukázka spuštění a běhu backendu se souborem text.xml (Do výpisu jsou přidána odřádkování pro přehlednost.):

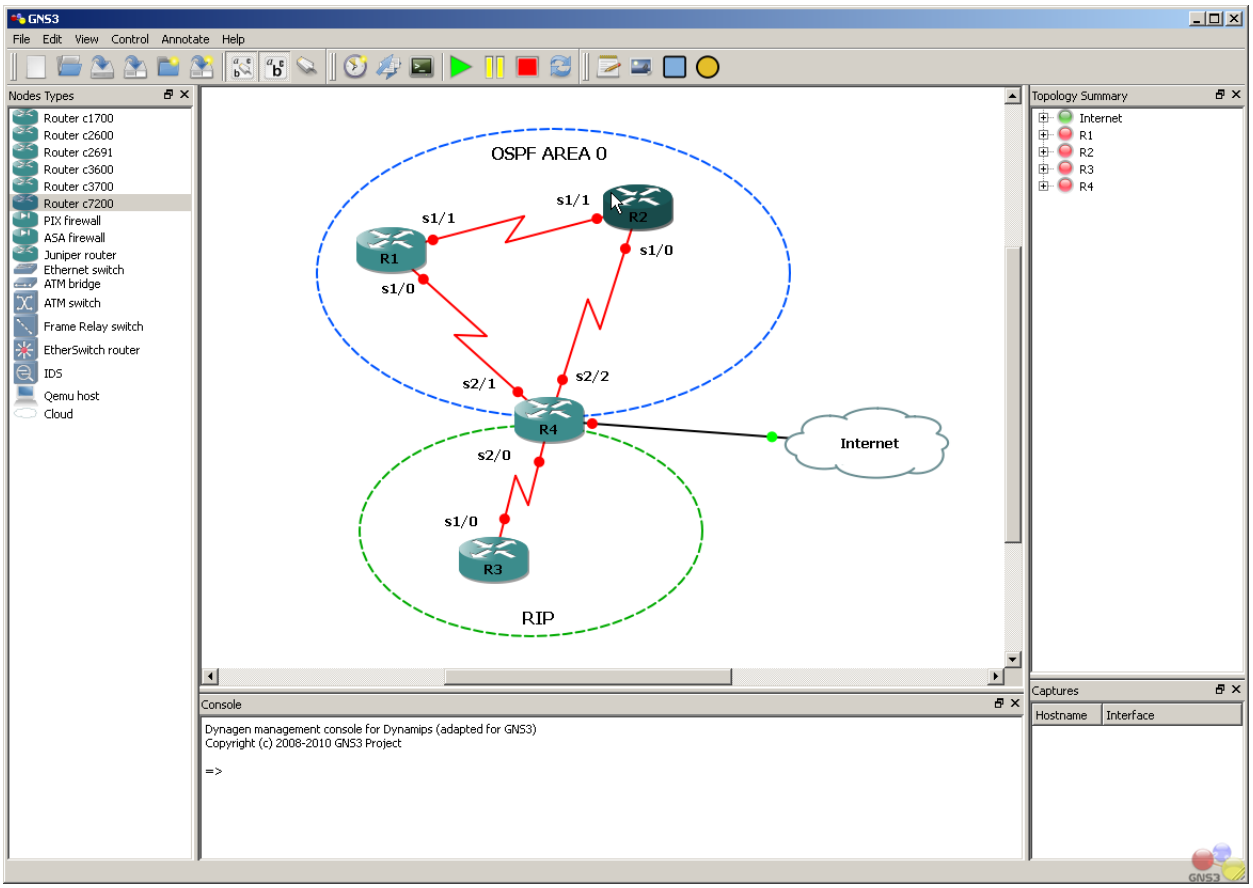
```
$ java -jar psimulator2_backend.jar /home/machvac/test.xml
Starting Psimulator2, build 2014-04-13
[IMPORTANT] TELNET: TELNET LISTENING PORT: : Device: pc1
```

[16]KVM – Kernel-based Virtual Machine. KVM je virtualizační infrastruktura pro Linuxové jádro, které je změněno na hypervizor.

[17]Cisco ASA – Cisco Adaptive Security Appliance. Cisco ASA je primárně bezpečnostní zařízení, které slouží k ochraně sítí všech velikostí. Klíčovými technologiemi, které integruje jsou například IPS, VPN a firewall. [2]

[18]Cisco PIX – Cisco Private Internet eXchange. Jde o dnes již zastaralé zařízení, které sloužilo především jako IP firewall nebo pro dynamický překlad IP adres. V roce 2005 představila společnost Cisco Systems novější řešení Cisco ASA, které poskytovalo většinu funkcionality nahrazeného zařízení.

[19]Cisco IPS – Cisco Intrusion Prevention System. Jedná se o obecné označení různých zařízení, která jsou schopna provádět analýzu procházející komunikace a reagovat na nimi aplikováním různých bezpečnostních pravidel. Ve většině případů se nejedná o zařízení určená pouze pro tento účel. [3]



Obrázek 2.1: Graphical network simulator 3, zdroj: [6]

```
listening port: 11000 (pc1)
[IMPORTANT] TELNET: TELNET LISTENING PORT: : Device: cisco2
listening port: 11001 (cisco1)
[IMPORTANT] NETWORK_MODEL_LOAD_SAVE: config.configTransformer.Loader:
Configuration succesfully loaded from: /home/machvac/test.xml
[IMPORTANT] EVENTS_SERVER: PACKET FLOW SERVER: :
Server sucessfully started, listening on port: 12000
[INFO] NET: pc1      IPLayer: IpPacket: src: 192.168.1.1
dst: 192.168.1.2 ttl: 64 IPv4 | IcmpPacket: REQUEST ZERO id: 1026 seq=1
| ARP reply received, sending packet to nextHop.
[INFO] NET: pc2      IPLayer: IpPacket: src: 192.168.1.1
dst: 192.168.1.2 ttl: 64 IPv4 | IcmpPacket: REQUEST ZERO id: 1026 seq=1
| Received IP packet destined to be mine.
[INFO] NET: pc2      IPLayer: IpPacket: src: 192.168.1.2
dst: 192.168.1.1 ttl: 64 IPv4 | IcmpPacket: REPLY   ZERO id: 1026 seq=1
```

```
| Sending packet.
[INFO] NET: pc1      IPLayer: IpPacket: src: 192.168.1.2
dst: 192.168.1.1 ttl: 64 IPv4 | IcmpPacket: REPLY   ZERO id: 1026 seq=1
| Received IP packet destined to be mine.
```

V samotném výpisu je patrných několik důležitých informací:

- Argument spuštěného JAR[20] archivu je soubor obsahující topologii – v ukázce test.xml.
- Porty transportní vrstvy, na kterých je možné se připojit na CLI rozhraní zařízení – v ukázce 11000 a 11001.
- Port transportní vrstvy, na kterém je možné se připojit z frontendu, aby bylo možné zobrazovat zachytávané pakety – v ukázce 12000.
- Dále jsou ve výpisu uvedeny informace o vlastní komunikaci mezi zařízeními a různá důležitá oznámení.

Podle mého názoru může být v některých případech takový výpis značně nepřehledný, z toho důvodu by podle mělo bylo vhodné do výpisu přidat časové známky pro všechny události.

Druhá verze simulátoru částečně podporuje propojení s reálnou sítí. Propojení je možné realizovat pouze na operačních systémech používajících jádro Linux, zároveň je třeba doinstalovat funkcionalitu pro virtuální distribuovaný ethernet. Virtuální distribuovaný ethernet přidává uživatelské rozhraní, které umožňuje vytvořit na hostitelském počítači síťová rozhraní, která jsou vzájemně propojena, a tak mezi nimi může probíhat komunikace.

Simulátor řeší propojení s vytvořenými virtuálními rozhraními pomocí knihovny jNetPcap. Jedná se o java wrapper (obal) pro libpcap nebo WinPcap, aby bylo možné multiplatformní použití. Knihovna dovoluje odesílat uživatelem definované rámce na určené rozhraní operačního systému. Protože se jedná o specializovanou knihovnu, umožňuje komunikaci už na úrovni linkové vrstvy.

Společně se simulátorem je dostupný i skript, který má uživateli usnadnit dosažení této funkcionality. Pro spuštění skriptu je nutné mít oprávnění super uživatele na hostitelském počítači. Podle návodu pro použití skriptu by měl být použit se třemi parametry. Prvním je klíčové slovo pair, které udává, že vytvořená rozhraní mají být po vytvoření propojena pomocí virtuálního distribuovaného ethernetu. Dalšími dvěma parametry jsou názvy vytvořených rozhraní, pro přehlednost jsou v návodu použity názvy sim0 a tap0. Sim0 představuje rozhraní spojené se simulátorem a tap0 představuje rozhraní používané operačním systémem.

[20] JAR – Java ARchive. JAR je souborový formát, který se běžně používá k distribuci programů a knihoven napsaných v jazyce Java.

Použití spolu se simulátorem je vymyšleno tak, že uživatel vytvoří dvě nová virtuální síťová rozhraní. Jedno bude používat v rámci operačního systému a bude přes něj probíhat komunikace do reálné sítě. Důležité je, aby obě rozhraní měla přiřazenou IP adresu a masku sítě a byla ve stejné podsíti. V případě, že některá z podmínek nebude splněna nebo v ní bude chyba, nelze předpokládát, jaký bude výsledek pokusu o komunikaci do reálné sítě. Druhé rozhraní by mělo být svázáno s aplikací. To je třeba provést pomocí speciálního příkazu na některém ze zařízení ve vytvořené topologii.

Toto řešení propojení s reálnou sítí bylo vybráno na základě analýzy v práci Tomáše Pitřince [14]. Dle této analýzy by nemělo být použito přímočaré řešení, které využije knihovny libpcap nebo WinPcap pro přímé odesílání paketů na fyzickém síťovém rozhraní. Autor práce uvádí několik důvodů proč by bylo nevhodné tento způsob řešení použít.

„Simulátor by zachytával všechny pakety přicházející do hostitelského počítače a odesílal pakety ze simulátoru. Tato možnost se však ukázala jako velmi nevyhovující z několika důvodů. Simulátor by v tomto případě zachytával veškerou síťovou komunikaci hostitelského počítače, což by ho velmi zatěžovalo. Druhou velkou nevýhodou je fakt, že při tomto řešení by nevzniklo logické propojení mezi hostitelským počítačem a virtuálním počítačem simulátoru, ale pouze mezi virtuálním počítačem a sousedním síťovým zařízením hostitelského počítače, jak je to zobrazeno na obrázku 4.10. Z toho vyplývá, že spojení mezi virtuálním a hostitelským počítačem nebude možné navázat, pokud hostitelský počítač není připojen k internetu.“[14]

Myslím si, že vznik logického spojení mezi virtuálním počítačem simulátoru a hostitelským počítačem, není nijak zásadní, vzhledem k tomu, čeho se rozšíření snaží docílit. Pokud chce uživatel propojit virtuální síť s fyzickou, měl by si být vědom toho, že je nutné, aby existovalo propojení jeho počítače na fyzické vrstvě s konkrétní sítí, která umožní komunikaci s vybraným cílem. V případě, že nebude uživatel na fyzické vrstvě připojen k žádné síti, nebude moci komunikace probíhat, což by podle mě jistě každý uživatel očekával. Nesplnění této podmínky znamená, že uživatel chce využít funkcionalitu pro komunikaci virtuální sítě s reálnou, ale zároveň není do žádné reálné sítě připojen, což je podle mého názoru nesmyslné.

Druhý problém, který autor citované práce zmiňuje, je zachytávání veškeré komunikace hostitelského počítače. Podle jeho názoru by mělo být hlavním důvodem, proč se vyhnout takovému přístupu, přílišné vytížení hostitelského počítače. Dle mých zkušeností programy jako Wireshark nebo TShark, které jsou nejčastěji použity právě pro zachytávání a analýzu komunikace, nevytěžují hostitelský počítač v žádné větší míře. V rámci svého studia jsem často používal Wireshark a ten i při několika stech příchozích paketech za vteřinu nevytěžoval můj počítač nijak výrazně. Zásadním rozdílem by ale mohl být implementační jazyk. V případě Wiresharku byl pro implementaci zvolen jazyk C, který dává programátorovi velké množství možností, je témeř vždy zárukou rychlosti a také nabízí mnoho možností optimalizace.

Pokud existuje možnost, jak tuto knihovnu využít a zároveň minimalizovat vytížení počítače, bylo by podle mého názoru vhodné takovou možnost analyzovat. V případě, že by se ukázalo, že vytížení počítače by nebylo nijak výrazné, bylo by podle mého názoru vhodné znovu zvážit realizaci právě pomocí tohoto řešení. Přestože implementace použití knihovny pro generování komunikace do reálné sítě by mohla být optimalizována, problémem by mohla být nutnost začlenění do existující implementace, která nemusí být optimalizována, nebo by neumožnila použití, které by nevytěžovalo hostitelský počítač pouze v malé míře.

Ačkoliv řešení ve druhé verzi simulátoru pracuje korektně a podle požadavků, podle mého názoru má několik nevýhod. Některé z těchto nevýhod je podle mého názoru možné vyřešit alespoň z části lépe, než jak tomu je v současném stavu aplikace. Bohužel některé nelze za žádných podmínek plně odstranit. Dále jsou popsány ty nevýhody, které považuji za nejzávažnější.

První nevýhodou, kterou považuji za zásadní, je nutnost doinstalovat virtuální distribuovaný ethernet. Podle mého názoru by měla být aplikace po získání schopná pracovat bez jakýchkoliv dalších zásahů do operačního systému. Ačkoliv to asi nebude prakticky možné, už jen z důvodu vlastního zadání práce, které stanovuje, že komunikace má probíhat už na úrovni linkové vrstvy. Pokud není na počítači, kde budeme chtít tuto funkcionalitu aplikace použít, dostupné rozhraní, které umožnuje komunikaci na úrovni linkové vrstvy, tak bude stejně nutné toto rozhraní doinstalovat.

Další velkou nevýhodou je fakt, že uživatel musí konfigurovat rozhraní, která nijak logicky nepatří do vlastní navržené topologie. Spolu se správnou konfigurací všech rozhraní ve virtuální síti, může být podle mého názoru velmi jednoduché se v takovém nastavení ztratit nebo udělat chybu. Hledání takové chyby může být velmi obtížné a časově náročné, a proto se mnohdy vyplatí zkusit nakonfigurovat propojení s reálnou sítí znovu a důsledněji.

V případě, že by propojení fungovalo transparentně nebo by uživatel pouze vybral výstupní síťové rozhraní, pomocí kterého by probíhala komunikace virtuální sítě s reálnou, mohlo by to podle mého názoru téměr úplně odstranit možné nekonzistence a chyby. Díky tomu, že by byla velká část této funkcionality před uživatelem skryta, nemohl by nijak ovlivnit její fungování a tak by bylo možné předejít většině potenciálních chyb.

Celkově se mi práce s řešením propojení reálné sítě zdála velmi neintuitivní a při samotné konfiguraci jsem narazil na různé problémy, které měly za výsledek nefunkčnost propojení s reálnout sítí. Proto se budu snažit v rámci analýzy a návrhu vytvořit řešení, které nebude takovými problémy trpět nebo se pokusím alespoň minimalizovat jejich dopad.

Při vypracování analýzy se budu snažit dodržet omezení pro projení s reálnou sítí, které definoval jeden z autorů druhé verze simulátoru [14]. Pokud to bude možné, tak se pokusím vytvořit takový návrh řešení, který s nimi nebude v rozporu. V případě, že by nebyla jiná možnost, jak řešení vytvořit a zároveň zachovat definovaná omezení, bude nutné tato omezení porušit, ale jsem

přesvědčen, že by to na výslednou aplikaci nemělo mít žádné velké negativní dopady.

Kapitola 3

Analýza a návrh

3.1 Analýza možností operačních systémů

Velké množství času jsem věnoval analýze operačního systému Microsoft Windows. Snažil jsem se především zmapovat možnosti, které systém poskytuje a z toho určit, zda by nebylo již zavedenou funkcionalitu v simulátoru jednoduchým způsobem převést i do tohoto operačního systému. V případě, že by to bylo možné, stačilo by pravděpodobně zajistit pouze různé verze skriptu pro různé operační systémy.

Nejprve jsem předpokládal, že vytvoření virtualních síťových rozhraní bude možné buď pomocí virtuálního distribuovaného ethernetu nebo jiného nástroje. Proto jsem zpočátku nezkoumal možnosti, které operační systém v tomto ohledu poskytuje.

Nejdříve jsem zkoumal možnosti instalace virtuálního distribuovaného ethernetu na Microsoft Windows, aby byly definované příkazy definované ve skriptu, přestože bylo zřejmé, že bude nutné ho upravit. Zajistit instalovaný virtuální distribuovaný ethernet bylo velmi obtížné. Na webových stránkách repozitáře projektu je uvedeno, že podporovanými operačními systémy jsou pouze systémy používající jádro FreeBSD nebo Linux.[5] Přesto jsem se pokusil celý projekt různými způsoby na Windows převést.

První možnost, která mě napadla, byla vzít zdrojové kódy zkompilované pod linuxovým kompilátorem a spustit je na Microsoft Windows. Bohužel tento pokus dopadl naprostým neúspěchem. Po spuštění libovolného vytvořeného binárního souboru, který je součástí výstupu kompilace celého projektu, došlo k chybě. Hledání informací o chybě ukázalo, že se jedná o chybu subsystému MS-DOS[21] . Vlastním obsahem chybového hlášení bylo to, že NTVDM[22]

[21]MS-DOS – Microsoft Disk Operating System. MS-DOS je operační systém společnosti Microsoft a byl nejvíce používaným operačním systémem na osobních počítačích v 80. a 90. letech minulého století.

[22]NTVDM – NT Virtual DOS machine / Virtual DOS machine. NTVDM je technologie, která umožňuje na moderních počítačích spouštět zastaralé DOS-ové a 16bitové programy.

CPU[23] narazil neplatnou instrukci. Veškeré informace, které se mi podařilo ohledně tohoto problému nashromáždit, nasvědčovaly tomu, že problém vznikl již v době kompilace a není možné ho bez opětovné kompilace odstranit.

Proto jsem svou další pozornost zaměřil na snahu kompilovat zdrojové kódy přímo ve Windows pomocí nástrojů MinGW a Cygwin. Protože při pokusu o kompilaci originálního zdrojového kódu došlo k chybě, začal jsem s analýzou kódu. Analýza ukázala, že celý projekt používá architekturu pro práci s deskriptory souborů, která je typická pro Linuxové operační systémy a alespoň tyto části by musely být přepsány, aby mohly zkompilované programy na Windows pracovat korektně. Přestože jsem se pokusil část některých modulů přepsat tak, aby bylo možné je na Windows zkompilovat, nepodařilo se mi vytvořit funkční řešení, které by poskytovalo stejnou funkcionalitu jako na podporovaných operačních systémech.

Funkcionalitu, kterou nabízí virtuální distribuovaný ethernet, se mi ani pomocí menších úprav nepovedlo na tomto operačním systému zkompilovat. Jsem přesvědčen, že by byly nutné rozsáhlé úpravy v téměř celém projektu a možná ani to by nezaručilo funkční řešení. Další adekvátně časově náročné způsoby, které by umožnili používání virtuálního distribuovaného ethernetu na Microsoft Windows, jsem nenalezl. Vzhledem k rozsahu zdrojového kódu celého projektu jsem se rozhodl analyzovat další možnosti, jak vytvořit propojení s reálnout sítí.

Protože bylo třeba zajistit možnost vytváření virtuálních síťových rozhraní, začal jsem zkoumat možnosti, které v tomto ohledu nabízí samotný operační systém. Pro vytváření jiných typů síťových rozhraní než je rozhraní typu loopback je třeba do operačního systému přidat rozšíření, které toto umožní. Projekt OpenVPN v tomto ohledu nabízí pravděpodobně nejlepší možnosti. S instalovaným doplňkem je možné vytvářet rozhraní typu TAP-Windows Adapter V9 nebo obdobné. Po instalaci dolňku nasvědčovalo vše tomu, že tato dílčí část celého problému by měla být vyřešena.

Při analýze možností operačního systému jsem narazil na nástroj netsh. Jedná se o CLI nástroj, který umožňuje konfigurovat síťová rozhraní. Protože rozhraní nástroje umožňuje jednoduchým způsobem jeho použití v dávkových souborech, byl program podle mého názoru ideálním kandidátem pro vytvoření virtuálních síťových rozhraní, které jsem do té doby musel vytvářet nebo modifikovat v grafickém uživatelském rozhraní. Takový přístup by mohl poskytnout uživateli spolu se skriptem potřebné rozhraní pro komunikaci do reálné sítě pomocí jednoho zadaného příkazu, což by mohlo být velmi efektivní.

Jako další bylo třeba se zaměřit na vlastní skript. Skript vyžadoval oprávnění super uživatele a instalovaný virtuální distribuovaný ethernet. Získat oprávnění super uživatele pro mě nebylo nijak obtížné. Dle mých zkušeností

[23]CPU – Central processing unit / Central processor unit. CPU je hadrwarová součást počítače, která vykonává instrukce počítačového programu.

má většina běžných uživatelů přihlášení na super uživatele zakázané pomocí registrů. Pro získání oprávnění super uživatele je nutné tyto hodnoty modifikovat. Po modifikaci je možné se přihlásit jako super uživatel a spustit požadované programy v priligovaném režimu.

Úkolem skriptu bylo vytvořit virtuální síťová rozhraní a propojit je. Použití předpokládá spuštění na operačních systémech s jádrem Linux, proto bylo třeba zjistit, jak něčeho takového docílit v operačním systému Microsoft Windows. Problém s vytvářením virtuálních rozhraní se mi již podařilo vyřešit, proto jsem se dále rozhodl analyzovat možnosti směrování v tomto operačním systému. Předpokládal jsem, že pomocí vhodného nastavení směrování bude možné docílit obdobné funkcionality, jakou nabízí propojení dvou rozhraní realizované virtuálním distribuovaným ethernetem.

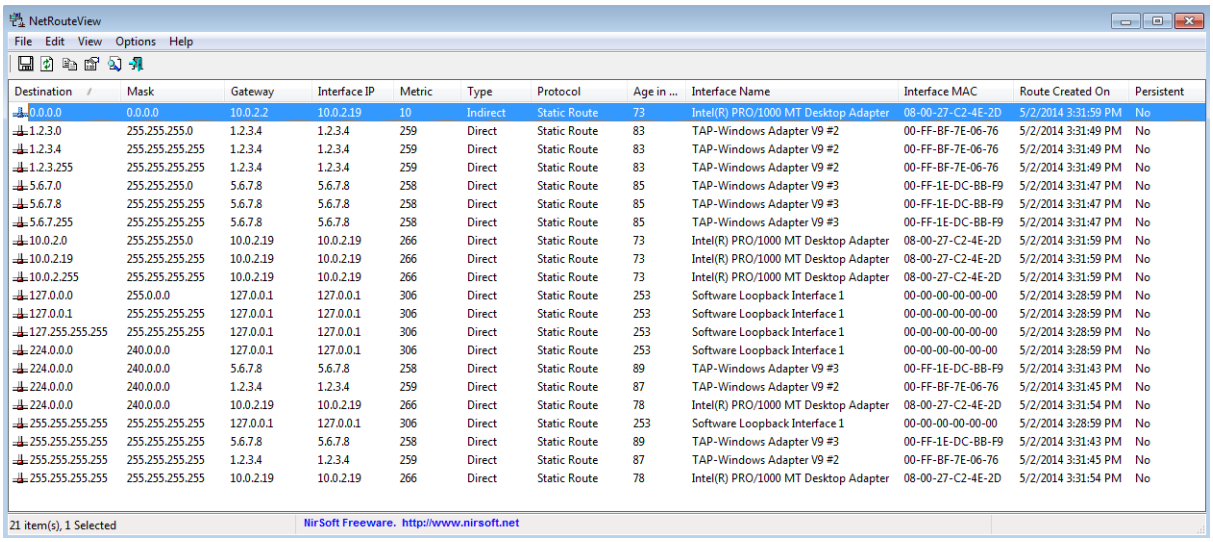
Pro manipulaci se směrovací tabulkou operačního systému je k dispozici v příkazovém interpretu cmd.exe příkaz route. Příkaz dovoluje uživateli přidat statické cesty, umí pracovat s verzemi 4 i 6 protokolu IP, navíc je možné definovat i metriku přidávané cesty. Bohužel podle mého názoru není syntaxe příkazu nijak intuitivní a jeho používání způsobuje mnoho chyb, před dosažením požadovaného výsledku. Směrovací tabulka modifikovaná tímto způsobem navíc přetrvává pouze od okamžiku modifikace do vypnutí nebo restartování operačního systému.

Pro pohodlnější manipulaci se směrovací tabulkou je pro Microsoft Windows dostupný program NetRouteView. Program není standardní součástí systému, ale je nutné ho doinstalovat. Jedná se o grafický nástroj, který poskytuje stejnou funkcionalitu jako příkaz route. Podle mého názoru je grafická varianta o mnoho příjemnější na používaní. Grafické rozhraní navíc implicitně zobrazuje mnohem více informací a umožňuje například ukládát vybrané položky pro opětovné použití.

Směrovácí záznamy jsou realizovány jinak, než je obvyklé. Pro jedno vytvořené rozhraní jsou implicitně přidány tři statické záznamy. První záznam je pro satmotnou síť, je definována uživatelem určená maska. Druhý záznam reprezentuje vlastní určenou adresu, maska sítě je v tomto případě definována pro všechny bity adresy. Poslední záznam reprezentuje broadcast sítě, stejně jako v předchozím případě je maska definována pro všechny bity adresy.

Běžnější reprezentace vytvoří pro jedno rozhraní pouze jeden záznam, kde definuje výstupní rozhraní, masku sítě a adresu sítě. Už jen z těchto informací je možné zajistit, aby se komunikace korektně dostala k určenému cíli, proto je pro mě záhadou, proč by měla mít realizace směrovacích záznamů jinou podobu. Další informací, která je nutná pro komunikaci mimo lokální síť, je brána sítě. Ta má však v této běžné reprezentaci stejnou formu jednoho směrovacího záznamu.

Aby mohl operační systém Microsoft Windows směrovat komunikaci mezi rozhraními je třeba modifikovat obsah registrů. Implicitně je tato funkcionalita zakázána, protože velká většina uživatelů ji pravděpodobně nevyužije. Bohužel ani s touto funkcionalitou se mi nepodařilo na lokálním počítači pro-

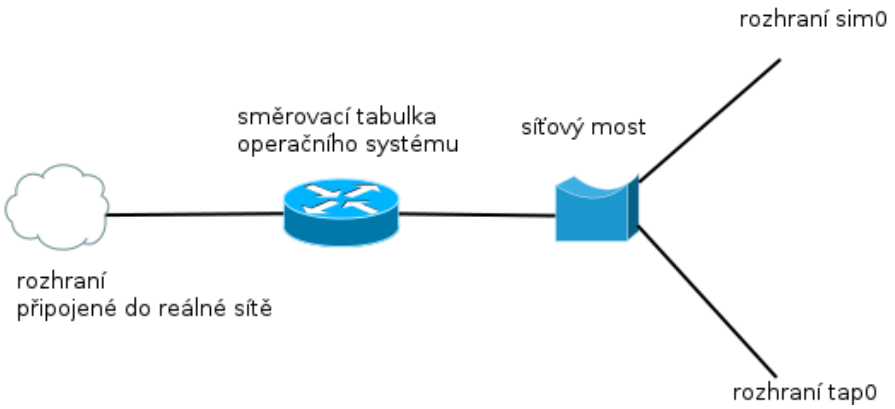


Obrázek 3.1: NetRouteView včetně virtuálních síťových rozhraní typu TAP-Windows Adapter V9

vést komunikaci mezi rozhraními. Přestože bylo možné komunikovat tam i zpět, problémem byla adresa, ze které jsem dostával odpovědi. Byla to adresa 127.0.0.1, která je pro člověka zabývajícího se síťovými technologiemi notoricky známá. Adresní rozhsah 127.0.0.0/8 představuje lokální počítač, nejčastěji se používá právě adresa 127.0.0.1.

Toto byl zásadní problém, který bylo třeba nějakým způsobem vyřešit. Pro řešení problému jsem se snažil analyzovat možnosti přemostění (bridging) síťových rozhraní, které Windows nabízí. Pokud jsou dvě nebo více síťových rozhraní přemostěna, znamená to vytvoření nového rozhraní, které reprezentuje všechna odpovídající rozhraní a rámce jsou mezi nimi přepínány na základě adres linkové vrstvy. Všechna rozhraní, která by měla být zapojena do propojení s reálnou sítí, by v této konfiguraci měla být součástí mostu. Statické záznamy každého rozhraní, které je přidáno do mostu, jsou automaticky odebrány. Z toho vyplývá, že směrovací tabulka neobsahuje žádné záznamy pro lokálně vytvořená rozhraní a operační systém neví, kam posílat pakety pro ně určené. Pro lepší představu o problému je přiložen obrázek 3.2, který má nastínit vnitřní topologii v operačním systému.

Bohužel v kontextu tohoto problému hrála velkou roli architektura operačního systému. Podle všech informací, které se mi podařilo v souvislosti s tímto problémem shromáždit, není možné způsob směrování v rámci takto vytvořených rozhraní ovlivnit požadovaným způsobem, aniž bych musel vytvořit program, který by definoval vlastní síťové rozhraní a vytvořil tak mezivrstvu mezi operačním systémem a samotnou aplikací. Vzhledem k časové náročnosti takového úkolu a různorodosti řešení pro různé operační systémy, které by



Obrázek 3.2: Schéma vnitřní topologie Microsoft Windows se dvěma virtuálními rozhraními zapojenými do síťového mostu

mohlo vzniknout, jsem tuto možnost zavrhnul.

Z této části analýzy vyplynulo, že pravděpodobně nebude žádným způsobem možné přenést již zavedenou funkcionalitu na operační systémy Microsoft Windows. Na problémy jsem narazil u všech částí, kterých se tato funkcionalita týkala. Dospěl jsem k názoru, že ani pomocí možností, které operační systém v daných částech řešení poskytuje, nelze zavedenou funkcionalitu na Microsoft Windows jednoduchým způsobem převést.

3.2 Analýza existujících řešení

GNS3 bylo rozsahem svých možností, funkcionalitou a volně dostupným zdrojovým kódem jasným kandidátem pro analýzu. Přestože by bylo možné rozebírat architekturu celé aplikace, zmíním pouze klíčové komponenty, které ovlivňují propojení s reálnou sítí. Kromě komponent pro práci se síťovými prvky jsou důležitou součástí hypervizory pro práci s virtuálními počítači.

Dvěma hlavními komponentami jsou Dynagen a Dynamips. Dynamips se stará o emulaci používaných obrazů operačních systémů Cisco IOS. Dynagen je textově orientované rozhraní pro Dynamips, které používá módu hypervizoru pro komunikaci s Dynamipsem. Pro mě bylo takové rozhraní zároveň se svou funkcionalitou něco, s čím jsem neměl výrazné zkušenosti. Pro představu přidávám následující demonstraci práce programu (Do výpisu jsou přidána odřádkování pro přehlednost. Veškerý vstup uživatele je uvozen znaky =>.):

```
$ dynagen /tmp/sample1.net
Reading configuration file...

NVRAM is empty, setting config register to 0x2142
```

```
C3725 instance
'ghost-c3725-adventerprisek9-mz.124-15.T13.bin-localhost.ghost' (id 3):
VM Status  : 0
RAM size   : 128 Mb
NVRAM size : 128 Kb
IOS image  : /tmp/c3725-adventerprisek9-mz.124-15.T13.bin
Loading ELF file '/tmp/c3725-adventerprisek9-mz.124-15.T13.bin'...
C3725 'ghost-c3725-adventerprisek9-mz.124-15.T13.bin-localhost.ghost':
starting simulation (CPU0 PC=0xffffffffbfc00000), JIT enabled.
C3725 'ghost-c3725-adventerprisek9-mz.124-15.T13.bin-localhost.ghost':
stopping simulation.
Network successfully loaded

=> list
Name       Type       State      Server          Console
R1         3725       stopped    localhost:7200  2001
=> start R1
C3725 instance 'R1' (id 0):
VM Status  : 0
RAM size   : 128 Mb
NVRAM size : 128 Kb
IOS image  : /tmp/c3725-adventerprisek9-mz.124-15.T13.bin

100-VM 'R1' started
```

Dynagen je psán v jazyce Python, stejně jako zbytek hlavních součástí GNS3. Jeho rozhraní dovoluje manipulovat se všemi zařízeními s operačním systémem Cisco IOS obsaženými v topologii. Argumentem programu je soubor s topologií. Ze zadané topologie program rozpozná zařízení v ní obsažená a na základě toho dovoluje případnou manipulaci s nimi.

Dynamips je psán v jazyce C, díky tomu nabízí velmi dobré možnosti téměř ve všech ohledech. Analýzu společně s Dynagenem jsem prováděl na jednoduché vytvořené topologii, která obsahovala jeden směrovač, který mohl pomocí oblaku (cloud) reprezentujícího hostitelský počítač komunikovat do reálné sítě. Tuto topologii jsem vytvořil v grafickém uživatelském rozhraní GNS3, ale dále jsem ho již nevyužíval, neboť pro další analýzy nemělo žádné potenciální přínosy.

Hlavními moduly Dynamipsu, které jsou zapojeny do komunikace s reálnou sítí jsou netio a vm. Z jejich analýzy vyplynulo, že pro propojení s reálnou sítí je použito řešení využívající knihovnu libpcap (WinPcap na Windows). Současná verze simulátoru Psimulator2 fungovala velmi podobně, navíc používala Java wrapper pro tyto knihovny. Zásadním rozdílem je to, že GNS3 posílá komunikaci virtuální sítě do reálné přímo pomocí rozhraní hostitelského

počítače. Psimulator2 toto obchází a vytváří mezivrstvu, kterou tvoří dvě vzájemně propojená virtuální rozhraní.

Důvod výběru takového řešení jsem detailněji rozebral ve druhé kapitole 2. Podle mého názoru jsou definovaná omezení, která vedla k výběru tohoto konkrétního řešení, nedostatečně opodstatněná. Problém, který by v tomto kontextu mohl podle mého názoru být opravdu závažný, je fakt, že aplikace by při použití tohoto řešení zachytávala všechny pakety určené pro hostitelský počítač.

Jak jsem již popsal v dřívějších částech práce, dle mých zkušeností není problém zachytávat i stovky paketů za sekundu, bohužel veškeré mé zkušenosti s takovými programy jsou specifické tím, že žádný z programů nebyl psán v Javě. Proto bylo obtížné zhodnotit, do jaké míry bude použití v Javě výkonově horší. Protože jsem neměl na základě analýz žádný jiný způsob vlastního řešení, mohl jsem pouze doufat, že implementace bude dostatečně efektivní pro obvyklé používaní. Ve výsledném řešení by pravděpodobně byla možnost zavedení optimalizace, ale zda by byla vzhledem k rozsahu projektu možná a jak by byla efektivní, to nebylo možné určit.

Vedoucím mi bylo doporučeno, abych prozkoumal i různé existující programy pro generování síťové komunikace. Na základě zkoumání několika multiplatformních programů jsem došel ke stejnému závěru, jako v případě GNS3. Všechny programy, které jsem v rámci této časti analýzy zkoumal, používaly pro generování komunikace knihovnu libpcap nebo WinPcap. To mě pouze ještě více utvrdilo v přesvědčení, že právě použití této knihovny bez další mezivrstvy (virtuální síťová rozhraní) by mělo být vhodným řešením.

Na základě těchto analýz jsem dospěl k názoru, že vlastní řešení propojení s reálnou sítí by mělo používat část již zavedené funkcionality, ale mělo by odebrat vytvořenou mezivrstu, která je tvořena virtuálním distribuovaným ethernetem. Tento krok by měl zaručit plnou portabilitu. Nejasná zůstala otázka, jak výkonné bude toto řešení spolu s použitím Java wrapperu pro knihovny libpcap a WinPcap.

3.3 Analýza a návrh části nového GUI

Na základě definovaného cíle, který stanovuje vytvoření lepšího návrhu propojení obou komponent, jsem vypracoval následující analýzu a návrh řešení problému. Při vypracování návrhu jsem se snažil zachovat všechny možnosti, které byly součástí původního řešení.

Současný stav grafického uživatelského rozhraní rozděluje frontend na dvě části – editor a simulátor. V editoru je možné upravovat vytvořenou topologii. V simulátoru je možné zobrazovat simulovanou komunikaci v reálném čase nebo ji pouze zachytávat k pozdějšímu zobrazení či analýze, taktéž je dostupná konfigurace všech zařízení pomocí vestavěného telnet rozhraní. Aby bylo možné vůbec zachytávat simulovanou komunikaci, je třeba propojit back-

end s frontedem. Toto propojení je realizováno pomocí tlačítka v části simulátoru.

Uživatel, který bude používat obě komponenty společně, pravděpodobně spustí nejdříve backend a poté frontend, kde bude chtít obě součásti propojit pomocí zmíněného tlačítka. Kliknutí na tlačítko pro projení vyvolá dialog, kam je třeba vyplnit parametry spojení. Dialog obsahuje předpokládané hodnoty, které jsou shodné s rozhraním frontendu. Hodnoty na straně backendu je možné zadat jako další parametry při sputění, pokud nejsou zadány, jsou použity implicitní hodnoty.

Backend automaticky volí porty transportní vrstvy pro zařízení od čísla 11000. Všechna zařízení obsažená v topologii obsadí v náhodném pořadí první neobsazené porty. Na prvním volném portu od čísla 12000 je vytvořeno spojení pro frontend, kde se přenáší informace o probíhající komunikaci, aby bylo možné je ve frontendu zobrazit.

Ačkoliv je tento způsob propojení funkční, mohou nastat potenciální problémy. Všechny problémové případy vyžadují, aby uživatel spustil nejdříve backend a poté už pracoval pouze ve frontendu. Pokud by chtěl uživatel takovýmto problémům předcházet, musel by vždy před simulací soubor uložit a spustit backend s aktuálním souborem. Tento krok nemusí být pro uživatele uživatele zřejmý, taktéž by uživatel mohl předpokládat, že změny budou schopné se promítnout do běžícího backendu.

V případě, že uživatel v editoru modifikuje existující topologii a pokusí se simulovat komunikaci v ní, nebude odpovídat obsah souboru backendu a uživatelem aktuálně otevřeného souboru. To může uživatele značně zmást. Nezáleží na tom, zda uživatel soubor po editaci uloží či nikoliv, pokud znovu nespustí backend s aktuálním souborem. Ve všech takových případech nastane nekonzistence buď mezi zobrazenou topologií nebo mezi načtenými soubory.

Z toho důvodu by podle mě bylo vhodné, aby k takové nekonzistenci nemohlo dojít a zároveň se pokusit v tomto kontextu řešit stanovený cíl – obě komponenty by se v případě potřeby měly jevit jako jedna. Podle mého názoru by mohlo být propojení komponent realizováno na podobném principu s pár drobnými úpravami. Pomocí menších úprav bych se pokusil vyřešit potenciální nekonzistence. Aby se obě komponenty mohly jevit jako jedna, bude pravděpodobně nutné spustit jednu z druhé.

Nekonzistencím bych se pokusil předejít sdílením topologie mezi oběma komponentami simulátoru. To znamená, že bude muset být zaručeno, že komponenta, která bude spuštěna jako druhá v pořadí, bude používat stejný soubor jako první komponenta. Zároveň bych znemožnil simulovat topologii, která není uložena. Tímto by mělo být zaručeno, že nebude možné dostat se do nekonzistentního stavu mezi soubory backendu a frontendu a zároveň bude zobrazená topologie v simulátoru odpovídat souboru. Tato opatření by měla zajistit, že se nikdy nebude možné dostat do nekonzistentního stavu mezi oběma komponentami.

Taktéž jsem se pokusil dbát na dostatečnou intuitivnost návrhu. Propojení

pomocí tlačítka mi připadalo jako vhodné řešení. Pokud by tlačítko nevyvolalo dialog, ale mělo jinou funkci, pomohlo by to podle mého názoru způsobu propojení komponent. Aby bylo možné odstranit nekonzistence, je třeba spustit jednu komponentu z druhé. Pokud by tlačítko realizovalo právě takovou funkci, bylo by pravděpodobně jednoduché zároveň předat soubor s topologií, který by samozřejmě musel být uložen.

Myslím si, že explicitní rozdělení simulátoru na dvě části pomocí dvou přepínacích tlačítek je zbytečné. Pokud by nebylo rozdělení navenek zřejmé, ale zůstalo by navržené tlačítko pro spuštění, dostal by se uživatel po kliknutí na toto tlačítko do části simulátoru. Současně se stiskem tlačítka by byl spuštěn backend s odpovídajícím souborem a parametry spojení. Jakmile by byl backend spuštěn a bylo by možné s ním z frontendu komunikovat, byl by uživatel automaticky přepnut do části simulátoru. Takové chování tlačítka by skrylo funkcionalitu přepínání obou částí, a tak by byl uživatel odstíněn od všech zbytečných detailů.

Po přepnutí do části simulátoru by se uživatel nacházel v původním rozhraní. Aby se uživatel mohl přepnout zpět do části editoru, bude třeba další tlačítko. Jelikož je zbytečné, aby zůstala možnost opětovného kliknutí na tlačítko pro spuštění backendu, nabízí se možnost toto tlačítko překrýt novým, které zajistí přepnutí zpět. Aby nebylo zbytečně plýtváno zdroji operačního systému, mělo by tlačítko zároveň s přepnutím zpět do módu simulátoru ukončit běžící backend. Takové chování by mělo dostat uživatele zpět do výchozího stavu, což by mělo zabránit jakýmkoliv potenciálním problémům.

Další částí tohoto návrhu by podle mého názoru měl být grafický prvek, který dovolí uživateli jednoduchým způsobem přistoupit k výstupu backendu. Výstup by měl být dostupný v reálném čase, aby uživatel mohl sledovat zároveň CLI rozhraní zařízení a výstup backendu obsahující informace o případné komunikaci. Myslím, že realizace pomocí tlačítka by měla být ideální.

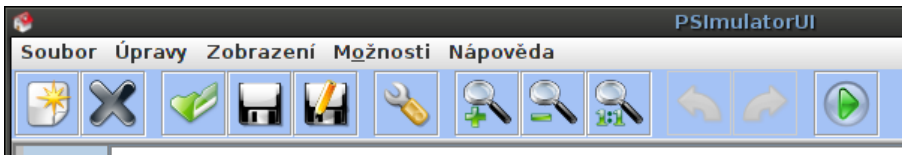
Výstup by bylo možné směrovat do některých částí vlastního simulátoru, ale vzhledem k množství textu, které backend produktuje by to podle mého názoru vedlo pouze k nepřehlednosti. Lepší variantou by podle mého názoru bylo stisknutí tlačítka vytvořit samostatné okno, kde by byl výstup dostupný.

Protože přímé směrování výstupu do simulátoru jsem zavrhnul v minulém odstavci, bude nutné použít nějaký mezikrok. Ideálním řešením je podle mého názoru přesměrování do souboru a následné čtení jeho obsahu. Takový návrh vyžaduje, aby byl výstup backendu ukládán do souboru na disku hostitelského počítače. Tento soubor by měl být vzhledem ke své povaze uložen v adresáři pro dočasné soubory. Pomocí přesměrování výstupu do souboru dosáhneme toho, že obsah souboru bude dostupný bez omezení, která by definoval emulátor terminálu v němž by byl backend spuštěn.

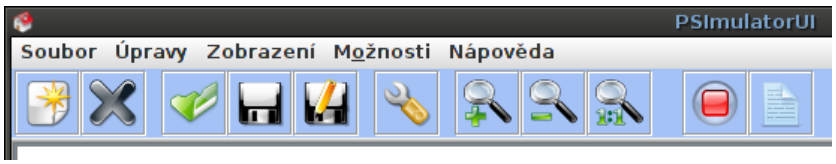
Okno, ve kterém by měl být zobrazen výstup backendu, by nemělo obsahovat žádné zbytečné grafické prvky. Podle mého názoru by mělo plně postačovat, kdyby okno obsahovalo pouze samotný text. Případně by mohlo být navíc schopno oddělit různé typy hlášení pomocí barevného zvýraznění. Okno

musí obsahovat posuvníky vzhledem k možnému rozsahu zobrazeného textu, taktéž by bylo vhodné, kdyby bylo možné změnit rozměry okna.

Výsledný návrh by podle předešlých analýz měl obsahovat jedno tlačítko v části editoru, které umožní spustit backend a automaticky přepne uživatele do části simulátoru. V části simulátoru bude původní rozhraní obohacené o tlačítko pro zobrazení výstupu backendu a tlačítko, které zastaví spuštěný backend a vrátí uživatele zpět do části editoru. V rámci návrhu jsem vytvořil obrázky 3.3 3.4, které ukazují jak by mohla potenciální realizace vypadat.



Obrázek 3.3: Nástrojová lišta včetně tlačítka pro zapnutí backendu (nejvíc vpravo)



Obrázek 3.4: Nástrojová lišta včetně tlačítek pro vypnutí backendu a zobrazení výstupu backendu (nejvíc vpravo)

3.4 Návrh řešení

Na základě provedených analýz jsem dospěl k názoru, že nové řešení by mělo používat velkou část původního, bude však nutné změnit několik částí rozhraní tak, aby bylo nové řešení konzistentní. Zásádní změnou by mělo být odebrání mezivrstvy tvořené virtuálním distribuovaným ethernetem. Tato změna znamená porušení omezení, které definoval autor původního řešení propojení s reálnou sítí [14].

Po odebrání této funkcionality bude nutné upravit rozhraní, které řeší samotné propojení s reálnou sítí. V aktuální verzi simulátoru je řešení tvořeno grafickým prvkem. Tento prvek by podle mého názoru měl ve svých vlastnostech dovolovat možnost výběru výstupního síťového rozhraní. Takové rozhraní by zaručilo, že bude postačovat, když uživatel provede konfiguraci pouze v grafickém rozhraní.

V případě, že by topologie obsahovala více prvků pro propojení s reálnou sítí, mohlo by dojít k různým nekonzistencím a zároveň by to pravděpodobně více zatížilo hostitelský počítač. Proto by bylo vhodné zavést omezení, které by definovalo, že maximální počet prvků pro propojení s reálnou sítí v topologii je roven jedné.

Další možností je takové omezení nedefinovat a řešit tento problém například pomocí stavu těchto prvků. To by znamelo pouze přesunutí omezení na úroveň počtu aktuálně aktivních prvků. Podle mého názoru je lepším řešením omezení na úrovni počtu prvků, ale bude záviset na již vytvořeném kódu, zda bude možné jednoduchým způsobem omezení přidat bez narušení zbytku programu.

Další potenciální nejasností je, zda by mělo být zachováno propojení rozhraní fyzického počítače se simulátorem speciálním příkazem rnetconn. Podle mého názoru by to nemuselo být explicitně nutné, avšak mohlo by to mít jisté potenciální přínosy.

Pokud by spojení s reálnou sítí výrazně vytěžovalo hostitelský počítač, dalo by se pomocí příkazu určit, zda bude propojení v daném okamžiku aktivní či nikoliv. To by dávalo uživateli možnost používat simulátor kontinuálně a spojení s reálnou sítí aktivovat pouze podle potřeby bez nutnosti změn v topologii.

Protože v této fázi práce nebylo zřejmé, jak bude realizace části nového rozhraní spolupracovat se zbytkem programu, rozhodl jsem se ponechat úvahu, zda by měl být příkaz pro aktivování propojení s reálnou sítí součástí výsledku, na implementační část. V implementační části by mělo být jasnější, zda bude vhodné ponechat tento příkaz jako součást řešení.

Protože implementace by měla pracovat s knihovnou jNetPcap, bude nutné zajistit, aby byla na hostitelském počítači dostupná. Stejně tak bude nutné, aby byly dostupné další knihovny, se kterými pracuje. Předpokládám, že toto bude podrobněji řešit instalační příručka.

Výsledek tohoto návrhu stanovoval pro uživatele následující postup:

1. Vytvořit topologii obsahující reálný počítač.
2. Vybrat ve vlastnostech reálného počítače výstupní rozhraní.
3. Uložit topologii.
4. Přejít do simulačního módu.
5. Na zařízení připojeném k reálnému počítači aktivovat propojení s reálnou sítí. (V této části ještě nebylo jasné, zda bude součástí finálního řešení.)

Po splnění všech kroků by se měl uživatel nacházet ve stavu, kdy bude moci vytvořená virtuální síť komunikovat s reálnou. Výsledek pokusu o takovou komunikaci bude samozřejmě záviset na konfiguraci obou sítí.

Kapitola 4

Realizace

Aby bylo možné rozšířit existující zdrojový kód aplikace, bylo nutné se nejprve seznámit s jeho strukturou. Protože jsem v jazyce Java dříve nepracoval na žádném tak rozsáhlém projektu, bylo pro mě zpočátku obtížné se v jeho struktuře orientovat. Projekt používá několik knihoven ve formě jar souborů. Vlastní zdrojové kódy jsou hierarchicky členěny dle Java balíků do složek. Výstupem sestavení zdrojových souborů jsou dva nezávislé soubory – jeden představující backend a druhý frontend.

Před zahájením vlastní úpravy zdrojových kódů bylo třeba zvolit vývojové prostředí. Vzhledem k rozsahu projektu nepřipadala v úvaha možnost, že bych nepoužil vývojové prostředí a pracoval pouze s libovolným editorem. Mezi nejčastěji používaná prostředí pro vývoj v jazyce Java patří NetBeans a Eclipse. Měl jsem dřívější zkušenosti s netbeans, ale nebyly nijak rozsáhlé a navíc mi toto prostředí připadalo zbytečně rozsáhlé pro mé dřívější potřeby. Na doporučení mých spolužáků, kteří měli s vývojem v Javě větší zkušenosti, jsem zvolil Eclipse.

Bohužel jsem byl nucem po nějakém čase své rozhodnutí změnit. Ačkoliv Eclipse je svým rozsahem i funkcemi srovnatelné s NetBeans, narazil jsem při práci na tomto projektu na několik problémů. Dospěl jsem k názoru, že jednodušší bude neztrácet čas řešením problémů, které podle shromážděných informací v NetBeans neměly nastat, a změnit vývojové prostředí.

Při práci jsem používal jednak vestavěnou dokumentaci Javy v NetBeans, která je automaticky zobrazena spolu se seznamem přípustných doplnění řetězce při stisku klávesové zkratky. Bohužel tato nápověda není vždy dostačující a proto jsem využíval i dokumentaci dostupnou na webových stránkách [11] [10]. Protože jsem používal knihovnu jNetPcap, bylo třeba pracovat i s její dokumentací [16].

4.1 Nové propojení komponent, úpravy GUI

Na základě definovaného cíle, který stanovuje vytvoření lepšího návrhu propojení komponent, jsem začal s realizací již vytvořeného návrhu. Nový návrh propojení obou komponent simulátoru byl zároveň spojen s novým návrhem části grafického uživatelského rozhraní. Protože zavedená funkcionalita pro přepínání obou částí simulátoru do jisté míry odpovídala mým záměrům, použil jsem již existující rozhraní.

Nejdříve jsem se zaměřil na vytvoření navžených tlačítek do nástrojové lišty. Nástrojová lišta je součástí hlavního okna aplikace, kde je realizována taktéž část jejího výkonného kódu, především reakce na signály od uživatele.

Nová funkcionalita, kterou bylo třeba vytvořit, bylo spustění backendu v samostnatném procesu po stisku tlačítka. Bylo několik možností, jak zajistit vytvoření nového procesu. Dva nejčastější způsoby realizace jsou pomocí rozhraní třídy Runtime.exec nebo ProcessBuilder. Existují ale i další možnosti, například Projekt Apache Tomcat [1] nabízí vlastní specializované rozhraní.

Pro potřeby simulátoru jsem vybral rozhraní třídy ProcessBuilder [13], protože mi její rozhraní připadalo nejvíce promyšlené ze všech standardních řešení. Pro další manipulaci s vlastním procesem je dostupná třídní metoda start, jejíž návratovou hodnotou je reference typu Process. Protože v implementaci řešení tohoto problému hrál významnou roli použitý operační systém, bylo třeba rozlišit spuštění na Windows a na Linuxu.

Na operačních systémech Microsoft Windows nebylo možné vytvořit externí proces přímo, ale bylo nutné ho spustit přes příkazový interpret cmd.exe. V dokumentaci příkazového interpretu cmd.exe [8] jsem našel informace o argumentech, které napomohly dokončit celé řešení. Aby se cmd.exe ukončil ihned po vykonání zadaného programu, bylo třeba použít argument /C.

Tento způsob použití způsobil, že při spuštění procesu se do proměnné, která reprezentovala běžící proces backendu, uložily informace o procesu cmd.exe. Očekával jsem, že signál k ukončení zaslaný programu, ho společně se všemi potomky ukončí.

Při snaze ukončit tento proces bohužel nedopadlo vše podle očekávání. To znamenalo, že všechny další potomci procesu byli ponecháni beze změn, přestože byl samotný proces ukončen. Ukončení procesu jsem prováděl pomocí metody destroy třídy Process [12]. Při hledání dalších informací o problému jsem narazil na různé doměnky, že se například jedná o bug samotného JVM.

Dokumentace metody uvádí: „Kills the subprocess. The subprocess represented by this Process object is forcibly terminated.“ [12] Taktéž je v záhlaví dokumentace této třídy uvedeno: „The methods that create processes may not work well for special processes on certain native platforms, such as native windowing processes, daemon processes, Win16/DOS processes on Microsoft Windows, or shell scripts.“ [12]

Z těchto informací jsem usoudil, že by bylo lepší se těmto problémům vyhnout. Protože jsem nenašel způsob, jak zajistit ukončení vytvořeného procesu

včetně potomků pomocí rozhraní Javy a reference na vytvořený proces, začal jsem analyzovat možnosti, které v tomto ohledu nabízí operační systém. Windows umožňuje ukončovat procesy pomocí grafického uživatelského rozhraní i pomocí CLI rozhraní. Dostupné CLI rozhraní tvoří programy tasklist a taskkill.

Vyšlo najevo, že toto byla správna cesta k úspěšnému dosažení cíle. Pomocí programu tasklist se mi podařilo v kódu vlastní aplikace získat PID[24] spuštěného procesu backendu. Pro získání PID backendu bylo třeba použít možnosti filtrování, které tasklist nabízí. Použil jsem filtr pro Image Name a Session Name.

Informaci o PID jsem dále využil a pomocí programu taskkill jsem byl schopen poslat signál k ukončení. Rozhraní programu taskkill vyžaduje některé další argumenty, aby mohlo pracovat s PID, ty jsem nalezl v dokumentaci [9]. Díky tomuto přístupu bylo možné ukončit spuštený backend. Po jeho ukončení se automaticky ukončí i cmd.exe, pomocí kterého byl backend spuštěn.

Úspěšné ukončení všech vytvořených procesů znamená návrat do výchozího stavu. Proto je tato funkcionalita navázána na tlačítko pro vypnutí. Návrat do výchozího stavu tedy znamená přechod zpět do módu simulátoru. Tlačítko pro vypnutí také smaže dočasně vytvořený soubor pro výstup backendu. Protože aplikace může být ukončena i v módu simulace, bylo třeba navázat funkcionalitu pro vypnutí běžícího backendu i na ukočení celé aplikace.

Na Linuxových operačních systémech je možné vytvořit externí proces bez nutnosti kaskády procesů jako v případě Windows. Proto jsem v samotném kódu použil způsob vytvoření procesu, který je obdobný spuštění programu na příkazové řádce:

```
./program.pripona
```

Tento způsob použití vyžaduje, aby operační systém věděl, pomocí jakého programu má program spustit. V případě, že se jedná o textový soubor, určí operační systém z hlavičky souboru, jak ho má vykonat. Pokud jde o binární soubor, určuje program přípona souboru. Nenašel jsem žádné informace, které by nasvědčovaly tomu, že by v některých linuxových operačních systémech neměl být soubor spouštěn pomocí JVM.

Protože nebylo možné dopředu znát parametry spojení obou částí, bylo nutné aby je frontend získal zpětně od spuštěného backendu. Díky přesměrování jeho výstupu do souboru to bylo jednoduše možné. Na základě takto předaných parametrů se frontend připojí k backendu a potom už je schopen zachytávat probíhající komunikaci.

Taktéž jsem přidal několik upozornění, která jsou zobrazena v případě, že uživatel před simulací modifikuje topologii a neuloží ji nebo topologie obsahuje

[24]PID – Process identifier. PID je číslo, které je unikátním identifikátorem spuštěného procesu v operačním systému.

prvek pro připojení do reálné sítě a je spuštěna s nedostatečnými oprávněními. První z nich zaručuje konzistenci simulované a zobrazované sítě. Druhé upozornění jsem přidal do případ, že uživatel zapomene spustit aplikaci v privilegovaném režimu.

Další součástí bylo tlačítko pro zobrazení výstupu backendu. Problémem bylo kontinuální zobrazování přibývajícího textu, které mělo okno v reálném čase zobrazovat. Bylo třeba použít takové řešení, které kontinuálně čte obsah souboru. Pomocí ukládání dočasné relativní pozice v souboru se toho dalo jednoduše docílit, bylo však nutné použít složitější rozhraní než takové, které se běžně v Javě pro čtení souborů používá.

Animaci, která byla součástí původního rozhraní pro přepínání obou komponent, jsem chtěl zachovat. Protože byla navázána na dialog s parametry spojení, který byl zobrazen ve vlastním okně, bylo její zachovaní vcelku problematické. Protože okno pro parametry spojení bylo v nově navrženém propojení zbytečné, bylo třeba ho odstranit, zároveň však animace vyžadovala pro zobrazení samostatné okno.

Rozhodl jsem se problém vyřešit pomocí okna s pevnou velikostí. (Rozměry původního okna byly dány velikostí vnitřních prvků.) Animace je zobrazena po spuštění procesu backendu, poté co jsou obě části schopné vzájemného propojení. Po skončení animace je uživatel automaticky přepnut do módu simulátoru. Myslím, že animace celkově vylepšuje dojem celé aplikace.

Původní rozhraní v části simulátoru obsahovalo tlačítko, které sloužilo pro odpojení od backendu. Toto tlačítko však sloužilo pouze pro odpojení, narozdíl od nově navrženého tlačítka, které má zároveň uživatele vrátit zpět do módu editoru. V novém návrhu bylo toto tlačítko zbytečné, proto jsem ho odebral.

Protože aplikace je vnitřně dostupná ve více jazykových verzích, bylo nutné se seznámit s konkrétní realizací této funkcionality. Aby bylo možné další rozšíření aplikace včetně lokalizace, byly v datové vrstvě aplikace, kde je umístěna hlavní část této funkcionlity, vytvořeny soubory pro jednotlivá jazyková prostředí. Soubory mají strukturu hodnoty a klíče. Díky této struktuře lze pohodlně ve zdrojových kódech referencovat potřebný klíč, který je při spuštění nahrazen konkrétní hodnotou. Konkrétní zobrazená hodnota je závislá na výběru jazykové verze v běžící aplikaci.

Aplikace taktéž umožňuje uživateli měnit velikost zobrazených ikon. Z toho důvodu bylo nutné všechny nově přidávané obrázky dodat ve všech požadovaných velikostech. Kromě vlastních vylepšení bylo vzhledem k provedeným změnám nutné provést revizi vestavěné nápovědy a zajistit její konzistenci.

Obsahem nápovědy byly snímky aplikace, které objasňovaly různé aspekty jejího používaní. Nápověda byla před úpravou rozčleněna do tří sekcí. Jednalo se o sekce tvorba sítě, simulační režim, pokročilé funkce. Vzhledem k provedeným úpravám nebylo třeba nijak měni vytvořenou strukturu, pouze bylo nutné dodat aktuální snímky rozhraní a upravit text.

4.2 Propojení s reálnou sítí

Realizaci této části řešení jsem začal úpravou vlastností grafického prvku představující reálný počítač. Na základě návrhu jsem vytvořil nabídku pro výběr rozhraní hostitelského počítače, které mělo sloužit pro připojení do reálné sítě. Toto řešení mi připadalo společně s popiskem pro uživatele dostatečně intuitivní.

Na Linuxu stačilo do nabídky přidat zařízení získaná pomocí rozhraní třídy Pcap, protože jejich názvy byly zároveň jejich identifikátory. Na Windows byla situace o poznání složitější. Stejný postup, pomocí kterého jsem na Linuxu získal seznam zařízení, obsahoval na Windows pouze registrové klíče jednotlivých zařízení.

Pomocí dalších metod třídy PcapIf, jejíž prvky byly zařízení obsažená v seznamu, jsem si zobrazil veškeré podrobnosti o každém ze zařízení. Metoda getDescription mi umožnila získat popis zařízení, které reprezentovalo vlastní síťové rozhraní. Zpočátku jsem předpokládal, že na Windows bude možné reprezentovat rozhraní právě pomocí této položky.

Protože při spuštění na Windows obsahoval seznam o poznání více zařízení, než která uživatel běžně používá, rozhodl jsem se je taktéž filtrovat. Popis zařízení tedy nemohl být určující pro přidaní zařízení do zobrazené nabídky, protože pak nabídka obsahovala i různá specifická zařízení jako například adaptér pro tunelové připojení isatap a podobné. Jako uživatel bych byl asi značně zmaten v případě seznamu obsahujícího taková zařízení při spuštění na počítači s jedním fyzickým ethernetovým rozhraním.

Pro účely filtrování jsem využil třídu NetworkInterface. Pomocí její metody getNetworkInterfaces jsem dostal seznam zařízení tak, jak je identifikuje operační systém. Na základě výstupů metod jsem byl schopen zařízení porovnat pomocí jejich popisu. Tento popis jsem využil a přidal jsem ho do nabídky, protože je podle mého názoru dostatečně vypovídající.

Vybrané rozhraní bylo třeba předat backendu, aby věděl s čím má dále pracovat. Problém byl v tom, že v době výběru rozhraní nebyl backend spuštěn. Předpokládal jsem, že po jeho spuštění, bude možné pomocí třídy Psimulator a metody getPsimulator nastavit identifikátor vybraného rozhraní.

Přestože jsem se snažil tohoto cíle dosáhnout různymi způsoby, všechny selhaly. Nabízelo se řešení, kdy by uživatel musel znovu v některé části backendu předat informaci o výstupním rozhraní.

Ve Windows by něco takového bylo jistě velmi problematické, protože nebylo možné říct, jaký konkrétní identifikátor by uživatel zvolil – zda například symbolické jméno zobrazené v operačním systému nebo identifikátor fyzického typu vlastního zařízení. Příkaz pro zadání informace mohl obsahovat nápovědu včetně příkladů, ale to by podle mého názoru situaci pouze zhoršilo.

Proto jsem se dále věnoval myšlence předání již vybraného rozhraní. Výsledné řešení pravděpodobně není zcela ideální, ale vzhledem k architektuře

simulátoru, by podle mého názoru bylo velmi problematické realizovat předání lepším způsobem, bez významných zásahů do architektury programu.

Vlastní řešení využívá dočasný soubor, ve kterém je předán identifikátor výstupního rozhraní. Pomocí společné identifikace souboru ve všech částech zdrojových kódů jsem byl schopen přenést úroveň tohoto problému z uživatele na zdrojový kód.

Omezení, která jsem definoval v části návrhu řešení propojení s reálnou sítí 3.4, se mi v rámci kódu grafického uživatelského rozhraní nepodařilo do programu vložit. Povedlo se mi definovat omezení trochu jiným způsobem, než jak jsem navrhnul. V případě, že uživatel umístil do topologie více reálných počítačů, všechny počítače mezi sebou sdílely vybrané rozhraní. Díky tomuto přístupu jsem zamezil veškerým potenciálním nekonzistencím.

Další částí řešení bylo rozhodnutí, zda má být zachován příkaz pro propojení s reálnou sítí na samotných zařízeních. Protože backend již měl k dispozici informaci o tom, jaké rozhraní má pro propojení použít, nebylo to nutné. Vzhledem ke dřívejšímu návrhu a celkové provázanosti s dalšími strukturami jsem se rozhodl příkaz ponechat a pouze modifikovat jeho rozhraní.

V případě, že bych vytvořil zcela nové rozhraní, mohl bych do programu zanést různé chyby. Taktéž bych tím znehodnotil veškerou práci dřívějších autorů. Přestože to nemusí být zřejmé, neznamenalo by to pouze znehodnocení dřívější implementace, ale především ztrátu veškerých výsledků testování samotné implementace. Něco takového by podle mého názoru mohlo mít významný negativní vliv na kvalitu výsledku, což byl taktéž hlavní důvod, proč jsem se rozhodl dříve vytvořené rozhraní zachovat.

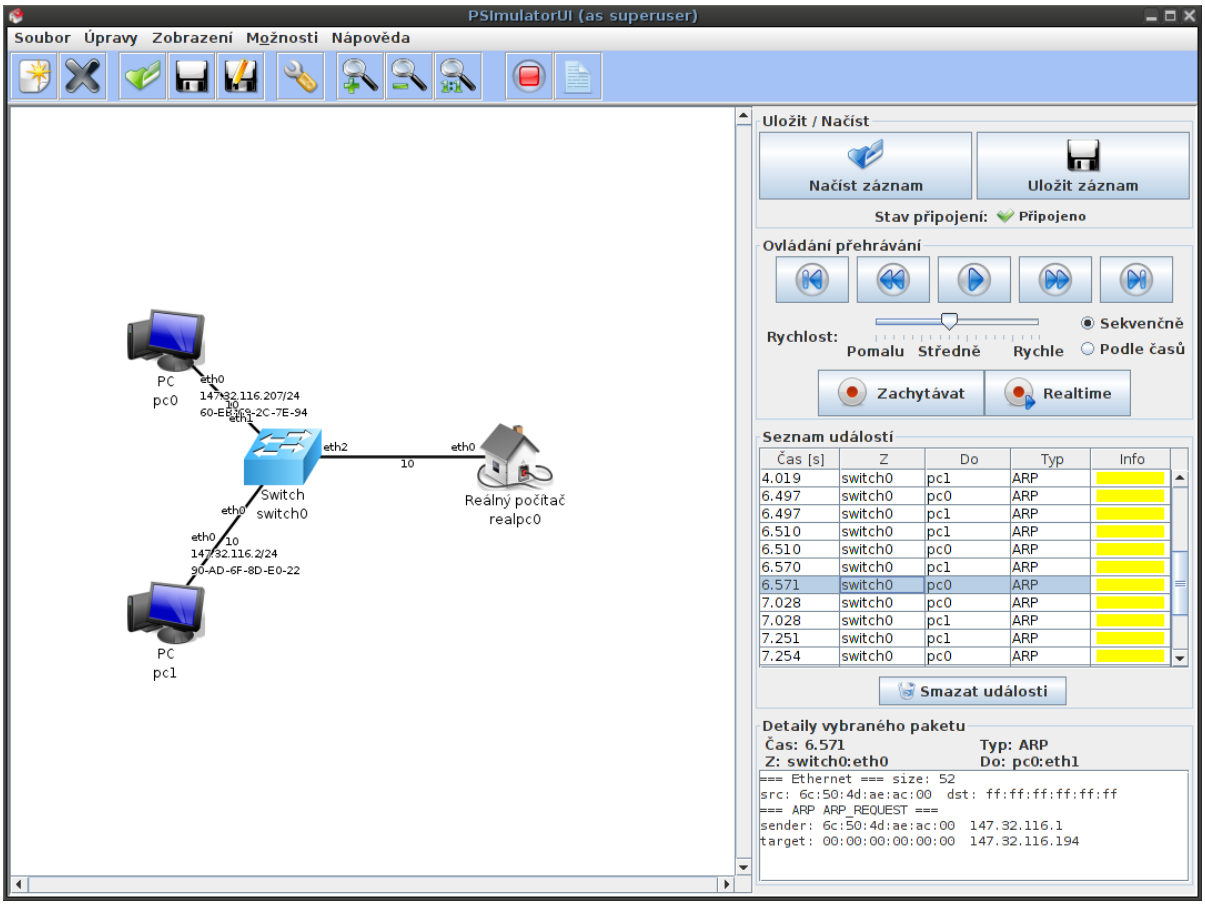
Abych odstínil uživatele od veškeré přebytečné konfigurace, snažil jsem se vytvořit nové rozhraní příkazu tak, aby bylo co nejjednodušší. Zachoval jsem název příkazu, ale upravil jsem dostupné argumenty, které je možné příkazu zadat. Pro maximální jednoduchost jsem definoval dva argumenty – *on* pro aktivaci a *off* pro deaktivaci propojení.

Pro spojení se zbytkem programu jsem využil již vytvořené rozhraní. Po úspěšném vytvoření funkčního řešení jsem byl schopen zodpovědět tíživou otázku, zda nebude řešení příliš vytěžovat hostitelský počítač.

Ukázalo se, že implementace je řádově stejně tak efektivní, jako v případě programů Wireshark nebo TShark. Pro ukázku je přiložen obrázek 4.1 zachycující komunikaci s reálnou sítí. Vytížení procesoru mého počítače při zachytávání této komunikace bylo zanedbatelné.

Výsledek takového konkrétního propojení tak dává obdobnou funkcionalitu jako například zmíněný Wireshark. Protože aplikace je vnitřně schopná rozeznat pouze několik určitých typů paketů, není řešení plnohodnotné, protože reálná síťová infrastruktura produkuje nepřeberné množství různých paketů.

Podle mého názoru tento způsob použití výborně demonstruje fungování reálné sítě společně s připojeným hostitelským počítačem. Spolu s možností vizualizovat pakety v reálném čase v navržené topologii to poskytuje uživateli jedinečnou zkušenost, která velmi dobře vystihuje fungování sítě.

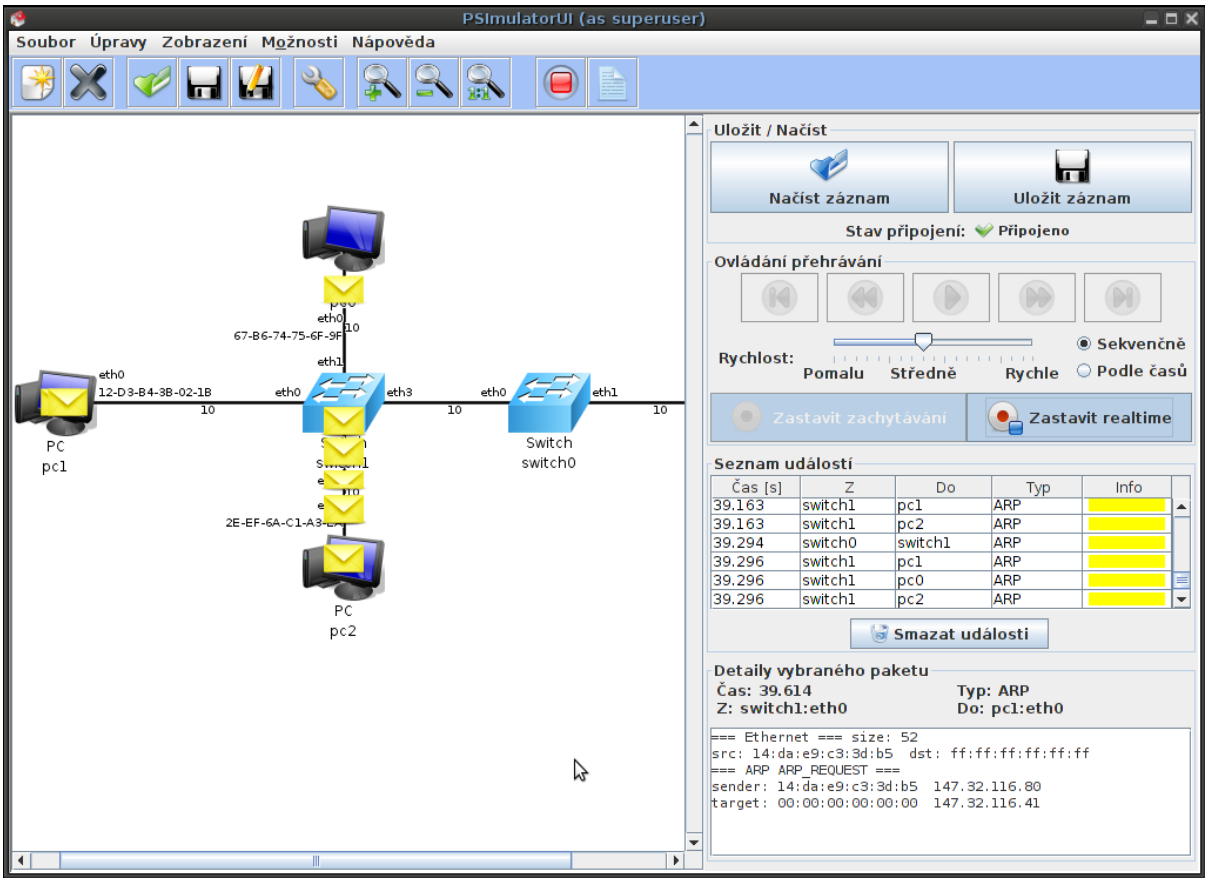


Obrázek 4.1: Ukázka zachycené komunikace s reálnou sítí

Protože vizualizace paketů v navržené topologii vytěžuje hostitelský počítač o poznání více, může kombinace s propojením s reálnou sítí způsobit výrazný nárůst vytížení počítače. Konkrétní stupeň vytížení bude v tomto případě dán počtem zobrazovaných paketů a jejich časovým odstupem. Pro ukázku je přiložen obrázek 4.2, který demonstruje popsanou situaci. Celkové vytížení procesu mého počítače dosahovalo okolo 90%.

Při testování přidávané funkcionality na Windows jsem odhalil, že v případě, kdy není knihovna jNetPcap instalována, není možné vyvolat okno s vlastnostmi reálného počítače. Přestože dojde k vyvolání vyjímky, lze simulátor dále používat, a tak není celková integrita aplikace ohrožena. Stejný problém by se pravděpodobně projevil na všech ostatních operačních systémech, v případě, že není knihovna instalována.

Aby bylo možné této chybě předcházet, hledal jsem způsoby, jak by bylo možné zkontrolovat instalaci knihovny před jejím vlastním spuštěním. Veškeré



Obrázek 4.2: Ukázka vizualizace zachytávané komunikace v reálném čase

informace, které jsem v souvislosti s tímto problémem shromáždil, se bohužel týkaly řešení problémů načítání nativních sdílených dynamických knihoven. Proto jsem se rozhodl v uživatelské příručce důrazně uvést knihovnu jako nutnost pro použití této funkcionality.

Výsledný způsob propojení je podle mého názoru velmi uspokojivý a díky své realizaci umožňuje propojit se s reálnou sítí pomocí libovolného rozhraní, které disponuje funkcionalitou požadovanou knihovnami. To dává uživateli možnost využít k propojení nejen ethernetová rozhraní. Bohužel realita je taková, že zařízení disponující požadovanou funkcionalitou jsou téměř výhradně ethernetová rozhraní.

Potenciální problémy, na které by mohl uživatel narazit při pokusu o komunikaci s reálnou sítí, jsou už záležitostí konfigurace obou sítí. Ačkoliv se může na první pohled zdát, že by tato komunikace měla fungovat ve všech případech, ve skutečnosti tomu zdaleka tak není.

Současná řešení metalických sítí velmi často definují různá omezení pro

koncová zařízení. Pokud se například uživatel na síti, kde je zavedena filtrace rámců na základě explicitně povolených adres linkové vrsty, pokusí o komunikaci z virtuální sítě, linková adresa komunikujícího zařízení pravděpodobně nebude obsažena v seznamu povolených adres, a proto bude výsledek negativní.

Podobných scénářů může na reálných sítích nastat mnoho v závislosti na stupni ochrany sítě, proto jsem se rozhodl úvest do uživatelské příručky informace o tom, jak nejlépe běžná opatření obejít. Obecně by se měl uživatel snažit, aby byla konfigurace simulovaného rozhraní, které bude komunikovat do reálné sítě, totožná s konfigurací reálného rozhraní.

4.3 Opravy chyb

Protože nejjednodušším problémem k řešení se mi zdálo logo, které se v některých případech spuštění aplikace zobrazí pouze z části, začal jsem tuto část práce právě u něj. Ukázalo se, že řešení tohoto problému není vůbec jednoduché.

Protože samotné logo není definováno nikde v kódu programu, nevěděl jsem, jakým způsobem tento problém řešit. Na základě rad od autorů dřívější verze programu jsem definoval zobrazení loga stejným způsobem, jak tomu bylo i v dřívější verzi programu.

Tento způsob využívá možnosti vývojového prostředí NetBeans pro přidání loga do výsledného sestaveného souboru. Přestože bylo možné definovat logo v samotném kódu aplikace pomocí třídy SplashScreen, rozhodl jsem se tuto možnost nevyužít, protože by to znamenalo zásah do kódu na nejvyšší úrovni. Hlavním důvodem pro toto rozhodnutí byly potenciální vzniklé chyby, které by podle mého názoru mohly přinést značně více problémů v porovnání s užitkem z korektně zobrazeného loga.

Dále jsem se pokusil odstranit různé problémy, které se v některých případech objevují při spuštění vestavěného emulátoru terminálu. Vlastní rozhraní je tvořeno několika desítkami tříd, proto nebylo vůbec snadné dohledat samotný původ problémů. Protože propojení tříd bylo nezvykle komplikované, rozhodl jsem se raději od řešení problémů upustit, abych do programu nezanesl další ještě výraznější chyby.

Úspěšně se mi podařilo odstranit další problém, který jsem popsal v první kapitole 1. Jednalo se o nevynucování přípony souborů. Pro práci se soubory používá program rozhraní třídy JFileChooser. Vhodnou kombinací se třídou FileNameExtensionFilter jsem dosáhl požadovaného výsledku.

Posledním problémem, který jsem chtěl ve výsledném řešení opravit, bylo sekvenční zpracování vizualizací arp paketů na přepínačích. Řešení tohoto problému, které jsem navrhnul, bohužel nebylo triviální a podle mého názoru by znamenalo významný zásah do celého programu. Z toho důvodu jsem od řešení raději upustil, protože jsem se obával případných negativních následků.

Kapitola 5

Testování

Vzhledem k provedeným úpravám jsem nepřišel na způsob testování, který by mohl být automatizovaný. Java nabízí možnosti jednotkového testování, to však nebylo podle mého názoru možné žádným způsobem využít. Z toho důvodu jsem se zaměřil na zkoumání vytížení počítače při spojení virtuální sítě s reálnou.

Možnost výrazného vytížení počítače, kterou jsem rozebíral v části realizace propojení s reálnou sítí 4.2, se ve výsledku nijak neprojevila. Zatížení počítače není nijak výrazné při desítkách příchozích paketů za sekundu na fyzické rozhraní počítače. Při zvýšení komunikace na stovky příchozích paketů za sekundu bylo již znatelné určité vytížení procesoru, stále se však nejednalo o výraznou zátež.

Nepodařilo se mi realizovat vetší datový tok, ale předpokládám, že při zatížení přibližně tisíce příchozích paketů za vteřinu už by zachytávání výraznou měrou zatěžovalo hostitelský počítač. Taktéž předpokládám, že uživatelé aplikace nebudou takové datové toky realizovat a proto by nemělo zatížení jejich počítačů působit žádné potíže.

Přestože jsem nepřišel na způsob jak automaticky otestovat vytvořený kód, snažil jsem se při jeho psaní vždy ověřovat správnost fungování přidávaných částí. Spolu s korektně pracujícím řešením na mém počítači, by podle mého názoru mělo být dostatečnou zárukou správně fungujícího kódu testování provedené s uživateli.

5.1 Testování s uživateli

S testováním výsledné práce s uživateli velmi pomohl vedoucí práce, který zajistil několik studentů předmětu BI-PSI. Testování se studenty jsme naplánovali na 6. 5. 2014. Motivací pro studenty byly body přislíbené za vyplnění připraveného dotazníku.

Pro testování bylo nutné, aby byla hotová uživatelská příručka. Společně s Michalem jsme takovou příručku vytvořili. Vzhledem k vestavěné nápovědě

v samotné aplikaci jsme se snažili, aby byla příručka ušetřena přílišných detailů. Protože nám připadalo zbytečné, aby byly oddělené instalační instrukce a instrukce pro používaní (v některých částech se prolínají), umístili jsme vše do uživatelské příručky.

Podle mého názoru by neměla být instalace vlastní aplikace nijak problematická. Rozhodli jsme se distribuovat vylepšenou aplikaci v zip archivu, stejně jako tomu bylo v případě původní verze. Stažený archiv stačí pouze rozbalit a dále už je možné používat aplikaci dle libosti.

Potenciální problémy by mohly nastat při instalaci knihovny potřebné pro propojení s reálnou sítí. Na webových stránkách projektu jNetPcap je popsán postup pro instalaci vlastní knihovny [15]. Bohužel návod na instalaci není podle mého názoru nijak názorný a proto si myslím, že by mnoho uživatelů mohlo při instalaci narazit na problémy.

Z toho důvodu jsem se rozhodl do instalační příručky přidat informace, které by měly instalaci uživateli usnadnit. Protože je možné získat různé verze knihovny pro různé operační systémy i architektury, rozhodl jsem se, že tato starost bude ponechána na uživateli. V případě, že by měla být knihovna součástí archivu, musel by obsahovat verze pro všechny podporované systémy i architektury.

Abychom měli z testování co nejlepší výstup a mohli ještě případně odstranit potenciální nedostatky, připravili jsme pro studenty dotazník. Spolu s ním bylo nutné připravit seznam úkolů a topologii, ve které měly být takové úkoly provedeny. Aby bylo otestováno maximum vytvořené funkcionality, snažili jsme se vytvořit topologii tak, aby prováděné úkoly proveřili co nejvíce z nových možností a aby zároveň byly nároky na objem činnosti uživatele co nejmenší.

Dále bylo samozřejmě nutné, aby byla k dispozici samotná aplikace. Pro verzování jsme používali SVN[25] na webovém repozitáři sourceforge.net [4]. Před testovaním jsme do společného projektu ve zmíněném repozitáři umístili archiv obsahující všechny potřebné materiály včetně jar souborů reprezentující obě hlavní komponenty simulátoru.

Testování probíhalo na dvou po sobě následujících cvičeních. Protože někteří studenti potřebovali odevdat zadané úlohy na předmět BI-PSI, zapojovali se do testování postupně. V průběhu testování jsme byli studentům k dispozici a v případě problému jsme se jim snažili poradit.

Na prvním cvičení narazilo hodně studentů na problém při snaze přejít do simulačního módu, přestože ani já ani Michal jsme na tento problém nenarazili. Studenti, kteří se s tímto konkrétním problémem setkali, používali vzájemně různé operační systémy. Tento problém se však neprojevil na žádné verzi Microsoft Windows.

Poradil jsem studentům, kteří se s problémem setkali, aby mi ukázali obsah dočasného souboru s výstupem backendu. Ukázka obsahu souboru z počítače,

[25]SVN – Apache Subversion. SVN je systém pro správu a verzování zdrojových kódů.

kde nastal problém (Do výpisu jsou přidána odřádkování pro přehlednost.):

```
/home/lukas/Downloads/PSI-simulator/psimulator2_backend.jar:
line 1: $'PK\003\004': command not found
/home/lukas/Downloads/PSI-simulator/psimulator2_backend.jar:
line 2: $'\b': command not found
/home/lukas/Downloads/PSI-simulator/psimulator2_backend.jar:
line 3: $'\b\035\260\245D\305': command not found
/home/lukas/Downloads/PSI-simulator/psimulator2_backend.jar:
line 4: uMETA-INF/MANIFEST.MFManifest-Version:: No such file or directory
/home/lukas/Downloads/PSI-simulator/psimulator2_backend.jar:
line 5: Ant-Version:: command not found
/home/lukas/Downloads/PSI-simulator/psimulator2_backend.jar:
line 6: syntax error near unexpected token `('
/home/lukas/Downloads/PSI-simulator/psimulator2_backend.jar:
line 6: `Created-By: 1.6.0_21-b07 (Sun Microsystems Inc.)
'
```

Na základě tohoto obsahu a způsobu spouštění backendu jsem došel k názoru, že operační systém nesprávně určuje implicitní program, kterým je backend spouštěn. Na operačních systémech používající jádro Linux mohou určovat implicitní program pro danou koncovku různé soubory – různí se napříč různými linuxovými distribucemi. Taktéž rozhoduje, zda je program spouštěn z grafického uživatelského rozhraní nebo z CLI rozhraní.

Protože různé systémy určují program, který má být spuštěn, různými způsoby, bylo třeba vymyslet dostatečně obecné řešení. Problém jsem se rozhodl vyřešit tak, že explicitně určím, že backend má být spuštěn pomocí programu *java* s přepínačem *-jar*. Taktéž jsem raději určil plnou cestu pro samotný program, abych předešel dalším potenciálním problémům. Díky systémové proměnné *JAVA_HOME*, která definuje instalační adresář Javy, by mělo být toto řešení plně portabilní.

Po opravení chyby jsem kontaktoval některé ze studentů, kteří na tento problém narazili. Potvrdili mi, že problém byl odstraněn. Protože pro tento přístup k vytoření procesu backendu nebylo nutné, aby byl soubor spustitelný (původní implementace to vyžadovala), odebral jsem taktéž kód, který postihoval tento stav.

Další problém, který testování odhalilo se týkal v podstatě toho samého, pouze s tím rozdílem, že se objevil pouze na Microsoft Windows. Šlo konkrétně o to, že při pokusu o přechod do simulačního módu zobrazil program chybové hlášení o tom, že soubor backendu nebyl nalezen.

Navržená instalační struktura vyžaduje, aby byly soubory backendu a frontendu umístěny ve stejné složce. Při rozbalení archivu tomu tak je, po-

kud však uživatel vytvořenou strukturu modifikuje, může na tento problém narazit.

Abych uživateli v chybovém hlášení více objasnil, jaký je problém a jak ho může vyřešit (v případě, že modifikoval vytvořenou strukturu), rozšířil jsem chybové hlášení. Zároveň jsem do uživatelské příručky přidal část textu o tom, že změny ve vytvořené struktuře mohou vést k chybám.

Další potenciální zdroj problému, který jsem v rámci testování odhalil byla cesta na souborovém systému, která obsahovala diakritiku. Bohužel si nejsem jist, zda byl toto jediný zdroj celého problému. Při snaze problém vyřešit, se mi ho povedlo úspěšně nasimulovat pomocí cesty obsahující diakritiku.

Z analýzy vyplynulo, že kód, pomocí kterého program získával umístění rozbaleného archivu, nebyl schopen pracovat s jinými než ASCII[26] znaky. Pro správné fungování bylo třeba použít metodu decode třídy URLDecoder. Po úpravě kódu se již problém znovu neprojevil.

Posledním výrazným problémem, který testování odhalilo, bylo zobrazení rozhraní ve vlastnostech zařízení. Při vyvolání vlastností nebyla vidět žádná rozhraní, po kliknutí na tlačítko pro přidání rozhraní a opětovném vyvolání okna byla rozhraní korektně zobrazena. Problém se projevil na Windows, Mac OS i na Linuxu, ale zároveň se s ním někteří studenti vůbec nesetkali, proto nic nenapovídalo tomu, čím by mohl být problém způsoben.

Bohužel jsme vůbec nepřišli na to, čím je problém způsobem, ani jak vznikl. Podle mého názoru musel existovat i v původní verzi programu, protože ani jeden z nás neprováděl modifikace částí kódu, které tuto funkcionalitu řeší. Další, avšak velmi nepravděpodobná, možnost byla, že problém byl způsoben různými verzemi JVM.

Rozhodli jsme se zkusit problém znovu nasimulovat a odstranit ho, bohužel ani jednomu z nás se nepodařilo ho znovu reprodukovat. Proto bylo téměř nemožné takový problém odstranit. Jako poslední možnost jsme zvažovali i kontakt autorů původní verze, ale vzhledem k charakteru problému, by pravděpodobně ladění zabralo mnoho času, proto jsme tuto možnost zavrhli.

Několik studentů se taktéž setkalo s problémem, kdy úspěšně nainstalovali knihovnu jNetPcap, ale chyběly jim vlastní knihovny nižší úrovně – libpcap nebo WinPcap. Proto jsem upravil instalační příručku, aby bylo explicitně uvedeno, že je taktéž nutné zajistit instalované prerekvizity. Taktéž jsem přidal nejnovější verze obou knihoven do archivu k aplikaci.

Vlastní testování mě svými výsledky vcelku překvapilo, protože velká část studentů se při testování simulátoru setkala s nějakými problémy, přestože já jsem při testování přidávané funkcionality ve výsledném kódu na žádné nenarazil. Nálezy chyb však byly pro vývoj aplikace pouze přínosem, protože jejich opravení přispělo ke zvýšení kvality aplikace.

[26] ASCII – American Standard Code for Information Interchange. ASCII je schéma pro kódování, které vychází ze znaků anglické abecedy a umožňuje kódovat 128 pozic.

Na druhém cvičení jsme do archivu přidali původní frontend, aby měli studenti, kteří by narazili na problém při přechodu do simulačního módu, možnost projít většinu úkolů. Taktéž jsme do nového frontendu přidali ladící výpisy, které nám měli pomoci odhadnout původ problémů.

Další věcí, kterou testování odhalilo, bylo to, že jNetPcap není v současné době dostupný pro operační systém Mac OS. Podle informací na webových strankách projektu [17] je podpora pro tento operační systém aktuálně vyvíjena. Přestože to znamená, že momentálně nelze použít zavedenou funkcionalitu na operačních systémech Mac OS, existuje možnost, že v budoucnosti to bude možné, aniž by musel být kód simulátoru změněn.

Testování s uživateli mě obohatilo o nové zkušenosti, které jsem do té doby vůbec neznal. Dříve jsem nevěděl, jakým způsobem navrhnout úkoly pro tento druh testování tak, aby byl jejich přínos pro tvůrce aplikace co největší. Taktéž mi dříve nebylo jasné, jakým způsobem do aplikace zakomponovat chybová hlášení tak, aby je mohl uživatel předat tvůrcům programu v případě nálezu chyby.

Z výsledků testovaní a reakcí studentů vyplynulo, že mnozí byly s aplikací spokojeni. Myslím si, že pozitivní reakce ve většině případů směřovaly k tomu, že by aplikace měla být studentům obecně schopná předat některé požadované informace nenásilnou formou. Někteří studenti se mě dokonce ptali, jestli by nebylo možné, aby zkusili aplikaci rozšířit o vlastní nápady.

Závěr

Úspěsně se mi podařilo rozšířit aplikaci o propojení s reálnou sítí, a tak splnit zadání bakalářské práce. Rozšíření je možné použít na operačních systémech Microsoft Windows i Linux. V současné době není tato funkcionalita dostupná na operačním systému Mac OS, protože knihovna realizující hlavní část funkcionality, tento operační systém prozatím nepodporuje.

Taktéž jsem do aplikace přidal další inovace, které nebyly v práce zadaní přímo vyžadovany, ale podle mého názoru pomohly celkově vylepšit možnosti pro uživatele simulátoru. Spolu s dřívější nabízenou funkcionalitou je podle mého názoru inovovaná verze simulátoru velmi dobrým nástrojem pro podporu výuky síťových technologií.

Jak jsem napsal již v úvodu práce, existují možnosti, jak aplikaci dále vylepšovat. Další rozšíření by mohli aplikaci ještě více obohatit, a tak ještě více rozšířit znalosti předávané uživatelům. Podle mého názoru si další potenciální pokračovatelé jistě najdou vhodné aspekty, které by mohly být rozšířeny nebo vytvořeny zcela nové.

Práce mi umožnila náhled do toho, jak by mohl probíhat vývoj softwaru v praxi a zároveň jsem si vyzkoušel práci v týmu. Dříve jsem neměl tak jasné představy o tom, co práce v týmu obnáší, ať už se jednalo o verzování zdrojových kódů, společnou tvorbu materiálů nebo komunikaci. Myslím si, že celkově mě práce obohatila takovým způsobem, který je pro budoucí praxi velmi přínosný. Taktéž jsem se seznámil s tím, jak probíhá práce na větším již existujícím projektu.

Věřím, že moje práce pomůže budoucím uživatelům při snaze o pochopení některých síťových technologií. Společně s prací Michala představuje výsledek unikátní nástroj, který kombinuje několik různých typů technologií a služeb, a tím je podle mého názoru na dobré cestě stát se obecně používaným.

Literatura

[1] Apache Software Foundation: Apache Tomcat - Welcome! [cit. 2014-05-08]. Dostupné z: http://tomcat.apache.org/

[2] Cisco Systems: Cisco Adaptive Security Appliance (ASA) Software - Products & Services - Cisco. [cit. 2014-05-02]. Dostupné z: http://www.cisco.com/c/en/us/products/security/adaptive-security-appliance-asa-software/index.html

[3] Cisco Systems: Intrusion Prevention System (IPS) - Cisco. [cit. 2014-05-02]. Dostupné z: http://www.cisco.com/c/en/us/products/security/intrusion-prevention-system-ips/index.html

[4] Dice: SourceForge - Download, Develop and Publish Free Open Source Software. [cit. 2014-05-05]. Dostupné z: https://sourceforge.net/

[5] Dice: VDE: Virtual Distributed Ethernet. [cit. 2014-04-29]. Dostupné z: https://sourceforge.net/projects/vde/

[6] GNS3: Graphical Network Simulator - GNS3. [cit. 2014-04-27]. Dostupné z: http://www.gns3.net/

[7] Horáček, M.: *Rozšíření síťového simulátoru o možnost použití konfiguračních souborů pro konfiguraci síťových prvků*. České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

[8] Microsoft Corporation: Microsoft Windows XP - Cmd. [cit. 2014-05-08]. Dostupné z: http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/cmd.mspx?mfr=true

[9] Microsoft Corporation: Microsoft Windows XP - Taskkill. [cit. 2014-05-08]. Dostupné z: http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/taskkill.mspx?mfr=true

[10] Oracle: Java Platform Standard Edition 7 Documentation. [cit. 2014-05-05]. Dostupné z: http://docs.oracle.com/javase/7/docs/

[11] Oracle: Java Platform, Standard Edition (Java SE) 8 Release 8. [cit. 2014-05-05]. Dostupné z: http://docs.oracle.com/javase/8/

[12] Oracle: Process (Java Platform SE 7). [cit. 2014-05-08]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/Process.html

[13] Oracle: ProcessBuilder (Java Platform SE 7). [cit. 2014-05-08]. Dostupné z: http://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html

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

[15] Sly Technologies Inc.: 1.4 - Installation | jNetPcap OpenSource. [cit. 2014-05-07]. Dostupné z: http://jnetpcap.com/node/328

[16] Sly Technologies Inc.: Documentation | jNetPcap OpenSource. [cit. 2014-05-05]. Dostupné z: http://jnetpcap.com/documentation

[17] Sly Technologies Inc.: What platforms are currently supported? | jNetPcap OpenSource. [cit. 2014-05-11]. Dostupné z: http://jnetpcap.com/node/270

[18] Švihlík, M.: *Vizualizace virtuální počítačové sítě.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, 2012.

Příloha A

Seznam použitých zkratek

API Application programming interface

ARP Address Resolution Protocol

ASA Adaptive Security Appliance

ASCII American Standard Code for Information Interchange

Cisco IOS Cisco Internetwork Operating System

CLI Command-line interface

CPU Central processing unit / Central processor unit

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DOS Disk Operating System

GNS3 Graphical network simulator 3

IP Internet Protocol

IPS Intrusion prevention system

ISO International Organization for Standardization

JAR Java Archive

JVM Java virtual machine

KVM Kernel-based Virtual Machine

LLDP Link Layer Discovery Protocol

MAC Media access control

MS-DOS Microsoft Disk Operating System

NTVDM NT Virtual DOS machine / Virtual DOS machine

OS Operating System

OSI Open Systems Interconnection

PDF Portable Document Format

PID Process identifier

PIX Private Internet exchange

SVN Apache Subversion

VPN Virtual private network

XML Extensible Markup Language

Příloha B

Instalační a uživatelská příručka

- Protože pokyny z obou sekcí se v některých částech prolínají, je vše společně umístěno v uživatelské příručce.
- Uživatelskou příručku lze najít na přiloženém CD v adresáři doc.
- Příručka je dostupná ve formátu PDF.
- Taktéž je k dispozici příručka v anglickém jazyce.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `doc` ..... adresář obsahující uživatelské příručky
- `examples` ..... adresář obsahující příklady použití
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Mach_Václav_2014.pdf` ..... text práce ve formátu PDF