UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Vizualizace topologie počítačové sítě

Bc. Lukáš Kuchta

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2011/2012**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: .....
Osobní číslo: .....
Studijní program: **N2646 Informační technologie**
Studijní obor: **Informační technologie**
Název tématu: .....
.....
Zadávající katedra: **Katedra informačních technologií**

Zásady pro vypracování:

.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

MURPHY, M.,L. Android 2 - Průvodce programováním mobilních aplikací. Brno: Computer Press, 2011. 371 s. EAN 9788025131947.
LEWIS, H. R., DENENBERG, L. Data structures and their algorithms. Berkley, Adison-Wesley, 1997.
Android developers Homepage [online]. 2011 [cit. 2011-10-07]. Dostupné z WWW: http://developer.android.com/.

Vedoucí diplomové práce: .....
Katedra softwarových technologií

Datum zadání diplomové práce: **31. října 2011**
Termín odevzdání diplomové práce: **18. května 2012**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2011

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 08.05. 2013 Lukáš Kuchta

Poděkování

Na tomto místě bych rád poděkoval svému vedoucímu diplomové práce Mgr. Tomáši Hudcovi za cenné rady, typografickou korekci, náměty a za odborné vedení během zpracování této práce.

Anotace

Práce pojednává o tvorbě vhodného nástroje, který bude mapovat a graficky zobrazovat topologii počítačové sítě. Na úvod jsou představena hotová řešení a diskutovány jejich nedostatky. Následně je představen souhrn technik, jimiž je topologii možné mapovat a vizualizovat.

Klíčová slova

graf, počítačová síť, kreslení grafu, algoritmy řízené silou, ICMP, UDP, vizualizace sítě, mapování topologie, JUNG2, JPcap

Title

Visualization of computer network topology

Annotation

The main goal of this master thesis is to create a tool for computer network topology visualization. The secondary goal is focused on the analysis of the existing visualization solutions and discussing their advantages and disadvantages. The thesis also introduces and analyzes a set of methods for topology mapping and visualization.

Keywords

graph, computer network, graph drawing, force-direct algorithm, ICMP, UDP, visualization of network, topology mapping, JUNG2, JPcap

Obsah

Seznam zkratek

UDP	User Datagram Protocol
TCP	Transmission Control Protocol
SNMP	Simple Network Management Protocol
ICMP	Internet Control Message Protocol
JUNG	Java Universal Network Graph
IP	Internet Protocol
TTL	Time to live
SLOC	Source lines of code
JNI	Java Native Interface
ORM	Object Rational Mapping
JRE	Java Runtime Environment
JDK	Java Development Kit
API	Application Programming Interface
MPLS	Multiprotocol Label Swtiching
RFC	Request for Comments

Seznam obrázků

Seznam tabulek

Seznam zdrojového kódu

1 Úvod

Složitost topologie počítačových sítí roste spolu s rozvojem nových technologií a vývojem mobilních zařízení jako jsou telefony, tablety a notebooky. Vývojový trend nebude v budoucnu zajisté upadat, nýbrž naopak (nasazení IPv6). Tato skutečnost tak sebou přináší vyšší nároky na orientaci v počítačových sítích. Pro zařízení jako jsou notebooky, telefony a podobné je typické připojení z různých míst proměnných v čase, na rozdíl od stolních počítačů. Udržet si tak přehled o aktuální topologii komplexnější sítě může být tedy poměrně náročné.

Podstata celé této diplomové práce spočívá v přiřazení grafické formy jinak abstraktnímu pojmu počítačová síť. Jinými slovy zviditelnit neviditelné.

V teoretické části je proveden rozbor dostupných aplikací řešících jistým způsobem daný problém nebo jeho vybranou část. Nástroje jsou podrobeny analýze vzhledem k možnostem vytvoření vhodného grafického modelu počítačové sítě, z nějž uživatel získá představu o skutečné topologii.

Dále jsou diskutovány technologie, jimiž je možné získávat informace o reálné topologii.

Závěr teoretické části je věnován přehledu možností, jimiž lze vizualizovat grafovou strukturu a bude poukázáno na problémy, které s touto oblastí souvisejí a jak je řešit.

Cílem praktické části je pak vytvořit aplikaci, která dává uživateli do rukou nástroj pro rychlou orientaci v síťové infrastruktuře a umožní mu zároveň udržet přehled o základní dostupnosti jednotlivých aktivních prvků zapojených do sítě. Aplikace bude orientována na malé až střední sítě se statickým směrováním a umožní reflexi skutečné sítě do podoby grafického modelu.

Aplikace bude nabízet uživatelskou volbu pomocí níž uživatel definuje příkazy, které skrze GUI vyvolá nad vybraným uzlem topologie.

2 Přehled dostupných nástrojů

Tato kapitola představuje rešerši dostupných nástrojů, které nabízejí možnost mapovat topologii počítačové sítě, zobrazovat její model a určitým způsobem s ním pracovat.

Jednotlivé produkty jsou členěny dle velikosti rozsahu sítě, pro kterou se jejich nasazení nejlépe hodí.

2.1 Topologie na úrovni velkých sítí

Jedná se víceméně o inteligentní softwarové balíky určené pro správu a řízení informačních technologií napomáhající korporacím konkurenčně obstát v dnešním nejistém byznys prostředí. Tyto balíky staví na principech modularity, čímž je možné rozšiřovat jejich funkcionalitu a postupně pokrýt individuální požadavky každého zákazníka. Vzhledem k jejich orientaci na správu a řízení služeb využívají ke své činnosti především protokol SNMP, který vyžaduje jistou aktivní spolupráci monitorovaných prvků. Software bývá nasazen v rámci enterprise sítí, jejichž hranice jsou obvykle ohraničeny firemní politikou, takže mají správci takovýchto celků na monitorované prvky obvykle úplný přístup.

Není neobvyklé, že taková síť dosahuje řádově tisíce prvků a její správa a dohled není triviální záležitostí. Nejsou tedy orientovány na pouhé mapování topologie, ale převážně na management a náročnou byznys sféru.

Mezi významné představitelé patří následující 4 komerční produkty od nadnárodních softwarových korporací:

- Tivoli od IBM,
- OpenView od společnosti Hewlett Packard,
- Spectrum od Computer Associates Technologies,
- Solslice Enterprise Manager od společnosti Sun/Oracle.

Mezi zástupce open source komunity se řadí tři významní kandidáti:

- OpenNMS,
- Nagios,
- Zenoss.

2.1.1 IBM Tivoli NetView

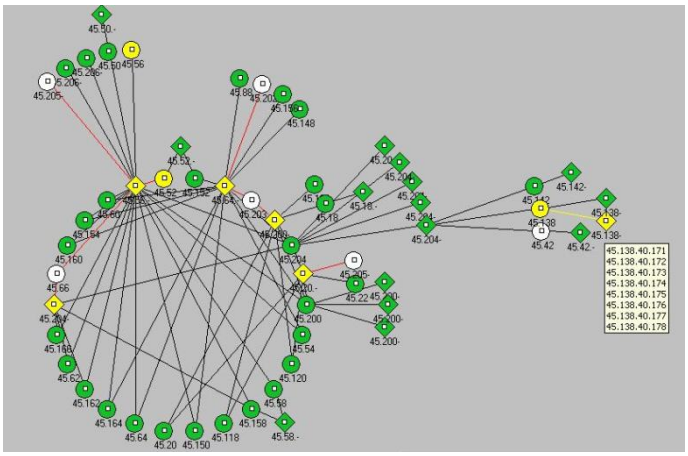
Jde o produkt společnosti IBM, který byl vyvíjen v ranních fázích společně se vývojovou divizí HP pod kódovým označením **OpenView**[1]. Po neshodách došlo k rozdělení a vývoj si řídila každá společnost samostatně. IBM začlenila OpenView do rodiny softwaru Tivoli a začala jej prodávat pod názvem NetView. Z hlediska stále rostoucích požadavků zákaz-

[1] Pod tímto názvem je produkt dále nabízen společností HP.

níků na software přestal ovšem tento produkt dostačovat. Tak se firma rozhodla postupně jej nahradit balíkem Netcool Precision.

Počátek vývoje celého komplexního balíku Tivoli byl zahájen firmou IBM roku 1996, poté co koupila společnost Tivoli System Inc. a zařadila tak její produkty do své nabídky. NetViev je postaven nad protokoly TCP/IP a poskytuje následující funkcionalitu:

- vizualizace topologie,
- monitoring dostupnosti prvků,
- sběr dat o síti, pomocí kterých je možné detekovat a predikovat problémy,
- propojení s analytickými nástroji z rodiny produktů Tivoli. [1], [2]



Obrázek 1 – topologická mapa IBM Tivoli NetView [3]

2.1.2 IBM Tivoli Network Manager

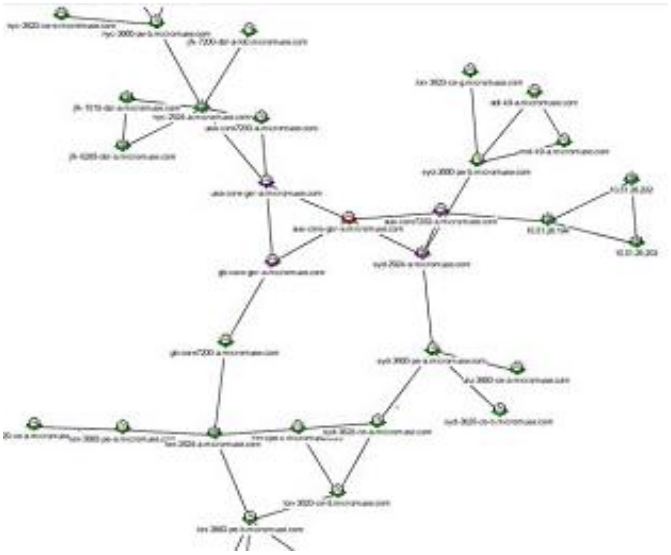
Označovaný rovněž jako IBM Tivoli Netcool/Precision, jak bylo zmíněno v předešlém odstavci, jde o náhradu produktu NetView. Nový produkt tak rozšiřuje jeho možnosti a to především v oblastech jako jsou automatická detekce infrastruktury a efektivní správa rozsáhlých počítačových sítí. Mezi hlavní výhody patří možnost vizuálně reprezentovat topologii a následně ji podrobit analýze s cílem maximalizovat spolehlivost a výkon sítě. Poskytuje odezvu téměř v reálném čase, což ho činí nepostradatelným pomocníkem při správě a tvorbě rozhodnutí u velkých sítí. Je orientován na sítě pracujících nad protokoly IP, Ethernet a MPLS. Dále nabízí možnost integrace s databázovými servery jako MySQL, Oracle Database a DB2, což otevírá dveře analytickým nástrojům pracujících s daty. Tento produkt je dostupný ve dvou verzích.

IP Edition – jedná se o analytický software pro komplexní správu rozsáhlých sítí, umožňuje diagnostiku a sběr příslušných dat. Dle výrobce poskytuje následující přednosti.

- Vizualizace topologie sítě s cílem umožnit rychlou orientaci v případě výskytu neočekávaných problémů.
- Uchovává data o fyzické a logické síťové struktuře a umožňuje jejich distribuci do řady specifických operačně orientovaných aplikací.

- Automaticky generuje mapy a pohotově reaguje na změny v topologii bez nutnosti ručních zásahů administrátora.
- Poskytuje přehled nevyužitých zařízení a portů, čímž umožňuje šetřit prostředky nutné na další rozvoj.
- Může pomoci vyřešit problémy ještě dříve, než k nim dojde na základě predikce z analyzovaných dat.
- Nabízí možnost automatizace procesů, což napomáhá šetřit čas a snižovat náklady na administrativní pracovníky. [4]

Transmission edition – jedná se o analogii k předchozí verzi s tím rozdílem, že tato je orientována pouze na lokální sítě.



Obrázek 2 – topologická mapa IBM Tivoli Network Manager IP Edition [5]

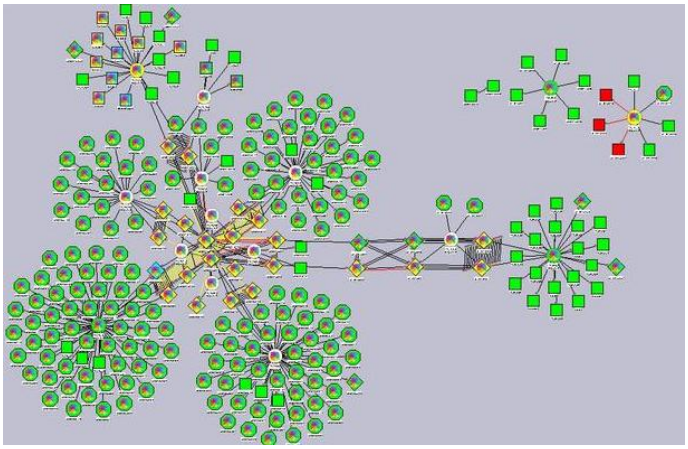
2.1.3 HP Network Node Manager

Tento softwarový produkt pochází z dílen společnosti HP a je součástí projektu HP OpenView (konkurenční produkt k rodině Tivoli od IBM). Zabývá se problémy v oblasti síťového a systémového řízení. Jeho cílem je zvýšit kvalitu síťové infrastruktury ve firmách, které se rozhodnou jej využívat. K dispozici je podpora pro databázové systémy PostgreSQL a Oracle Database. V podstatě nabízí podobnou funkcionalitu jako dříve popsaný konkurenční produkt výběrově.

- Automatická detekce celé síťové infrastruktury.
- Upozornění na změny, či problémy.
- Vytváření reportů z nasbíraných dat.
- Možnost integrace různých aplikací a další.

Produkt je dostupný rovněž ve více verzích, nicméně HP volila jinou politiku (u konkurenčního produktu IBM se nepodařilo demo verzi dohledat) a tak nabízí licence

omezené na počet sledovaných uzlů. Navíc je k dispozici, omezená (max. 250 uzlů) demo verze[2] pro případné zájemce k vyzkoušení zdarma. [6], [7]



Obrázek 3 – topologická mapa HP OpenView Network Node Manager [8]

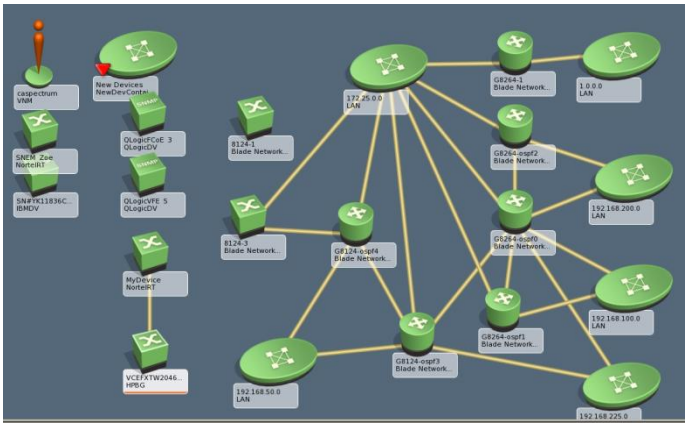
2.1.4 CA inc. Spectrum Infrastructure Manager

Produkt, který je dílem[3] společnosti Computer Associates International umožňuje automatickou detekci infrastruktury, její modelování a správu konfigurace. Jako u konkurenčních produktů je cílem maximalizovat spolehlivost IT služeb. Tento softwarový produkt je vhodný k nasazení ve společnostech, jež mají rozsah vládních organizací a správních celků. Obsahuje široké množství předdefinovaných stavů možných příčin poruch a způsobů pro jejich rychlé řešení, což výrazně urychluje dobu případných oprav. Systém se vyznačuje především velkou robustností, škálovatelností a velmi rozsáhlými možnostmi přizpůsobení. Ostatně jako dříve představené projekty poskytuje například:

- systém správy výstrah, který automatizuje proces zpracování problémových hlášení.
- Rychlou identifikaci a následnou nápravu chyb.
- Podrobně sleduje vliv stavu a výkonu sítě na dostupnost a úroveň služeb.
- Integrace se širokým spektrem produktů od jiných dodavatelů. [9], [10]

[2] Pro operační systémy Microsoft Windows pouze 64b verze, nebylo tedy možné vyzkoušet vzhledem k vlastnictví pouze 32b OS.

[3] Ve skutečnosti, byl projekt zahájen již roku 1991, firmou Cabletron Systéme. CA inc. koupila divizi zabývající se vývojem Spectra roku 2005.

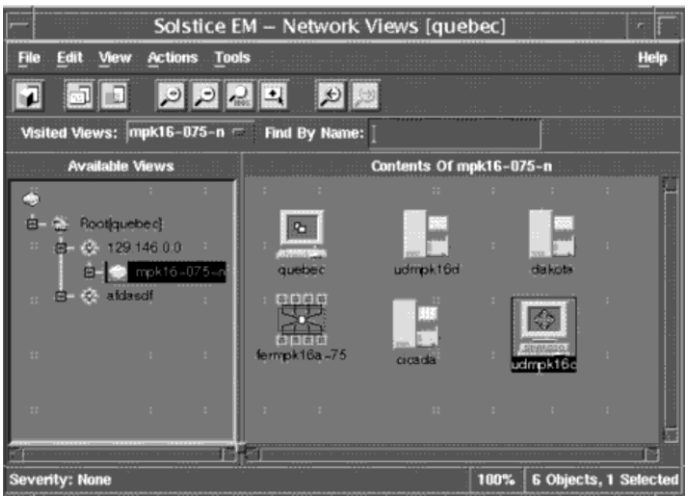


Obrázek 4 – topologická mapa CA Technologies Spectrum [11]

2.1.5 Sun Solstice Enterprise Manager

Opět jako předešlé projekty nabízí sadu nástrojů pro monitorování, vyhodnocování a zdokonalování sítě. Dále pak je zde možnost využít programové API (C++, Java) pro tvorbu vlastních řešení šitých na míru a přesně tak uspokojit ty nejnáročnější potřeby, což poskytuje vysokou flexibilitu a škálovatelnost. Je tedy možné použít stávající hotové nástroje a kombinovat s vlastním řešením. Dle společnosti Sun zvládá Sofistice EM spravovat až stovky tisíc uzlů. Sofistice EM umožňuje například:

- automatizovat objevování prvků v síti a sledovat tak měnící se topologii.
- Získávat informace o patřičných síťových prvcích, podpora SNMP, CNMP, RPC.
- Správa a konfigurace jednotlivých komponent a mnohé další.



Obrázek 5 – ukázka využití Java API [12]

Tento produkt je zde uveden spíše ze zajímavosti, protože Oracle[4] se rozhodl ukončit jeho vývoj včetně podpory a produkt byl roku 2007 stažen. V roce 2012 byla ukončena pak i předplacena podpora pro finální verzi 6.0.

[4] Koupil společnost Sun v roce 2000 se všemi jejich projekty.

2.1.6 OpenNMS

Open sourcový projekt zabývající se managementem enterprise sítí. Počátek vývoje zahájila společnost PlatformWorks roku 1999. Jako stabilní základ pro svůj vývoj volili autoři programovou platformu Java v kombinaci s databázovými projekty PostgeSQL a JRobin[5]. Vzhledem k použitému rámci Hibernate je ale vytvořena jistá databázová nezávislost díky ORM. Liší se oproti svým komerčním rivalům především tím, že poskytuje možnost stažení zdrojových kódů, což dává technicky zdatným uživatelům do rukou možnost dodefinovat potřebnou funkcionalitu, která není v základu dostupná, případně rozšířit stávající.

Ke stažení jsou zdarma bez licenčních poplatků dvě vývojové větve označované jako **stable** a **unstable**. Dále je pro případné zájemce možnost využít placenou nebo komunitní podporu. [13], [14]

Obrázek 6 – topologická mapa OpenNMS [15]

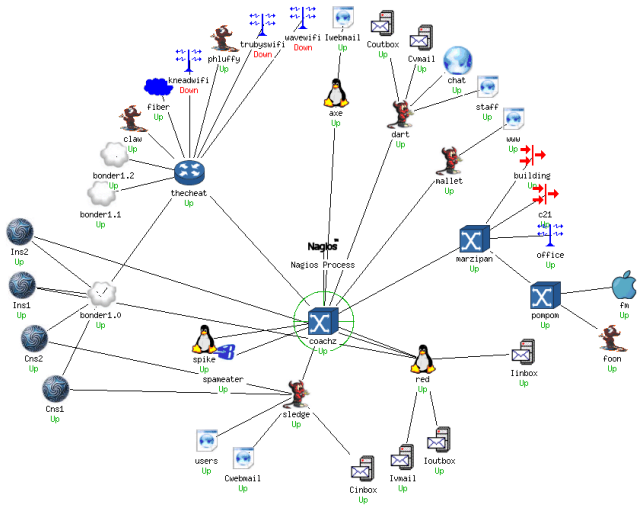
2.1.7 Nagios

Rovněž populární open sourcový projekt, jehož první veřejná verze se datuje od roku 1999 (původní označení projektu NetSaint). Jádro systému je vytvořeno v jazyce C a pro data je možnost využít flat[6] soubor, případně SQL databázi. Nagios umožňuje dozorovat síťovou infrastrukturu, servery a služby. Je dostupný ve dvou verzích **Open Source** a **XI.** Zásadní rozdíl je v tom, že XI verze je zaměřená na enterprise třídu a je tedy placená, přičemž exis-

[5]Port nástroje RRDTool pro jazyk Java umožňující uchovávat datové kolekce proměnné v čase.

[6] Jednoduchá databáze ve formě tabulky uložená v textovém souboru. Data jsou reprezentována prostým textem.

tuje její omezená verze zdarma, mající limit na sledování 7 hostů. Verze Open Source je k dispozici zdarma bez omezení.



Obrázek 7 – topologická mapa Nagios [16]

Za povšimnutí stojí skutečnost, že jistá rivalita existuje i mezi zástupci OSS komunit. Počátek vývoje obou projektu byl zahájen ve stejném roce, takže mají zhruba stejné zkušenosti co se vývoje a technologických podmínek týče.

Na stránkách produktu Nagios je k dispozici elaborát, který porovnává oba výše diskutované projekty a Nagios, zde vítězí bez sebemenší ztráty. Splňuje veškeré požadavky definované autory na 100 %, OpenNMS na první pohled poměrně dost zaostává. Naproti tomu na stránkách OpenNMS je pouze zmínka o tom, že Nagios není v žádném případě plnohodnotnou náhradou za aplikaci sloužící ke správě enterprise sítí. Autoři jdou ještě o kus dále a staví svůj produkt na úroveň HP OpenView, který je jeden z prvních svého druhu[7]. Dále pak uvádějí zmínku o tom, že Nagios nebyl od počátku vyvíjen jako projekt pro byznys třídu a že jejich produkt se drží podnikových zásad od počátku vývoje, nicméně politika Nagiosu působí více útočněji. Faktické rozdíly jsou následující:

- jazyk: jádro Nagiosu je psáno v jazyce C naproti tomu ONMS v Javě.
- Objevování uzlů a služeb: u Nagiosu nutno explicitně určit, ONMS obsahuje vyhledávací algoritmus. [17], [18]

2.1.8 Zenoss

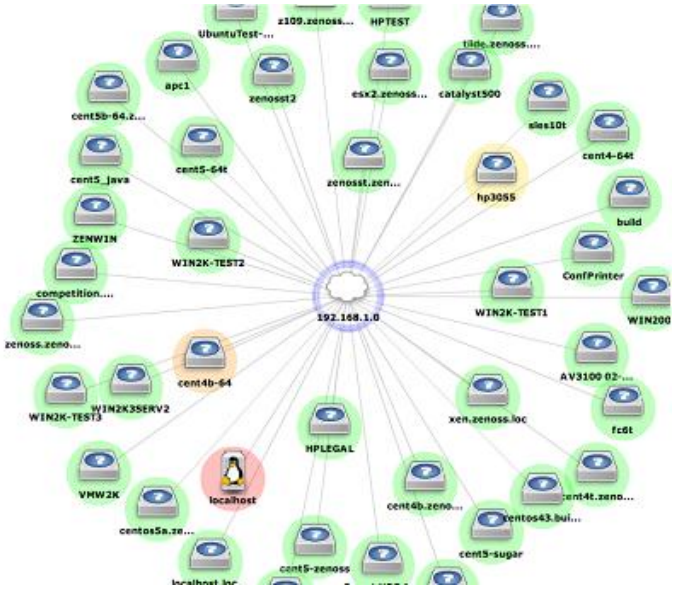
Nejmladší zástupce OSS, jehož vývoj začal roku 2002 a staví na základech jazyka Python v kombinaci s MySQL, RRDTool, ZODB[8]. Jedná se o inteligentní nástroj sloužící k monitoringu a řízení služeb a serverů. Stejně jako jiní nabízí integraci s informačními

[7] V současné době se podle údajů HP používá ke správě až 70 procent všech zařízení na Internetu.

[8] Zope Object Databáze – objektově orientovaná databáze sloužící pro persistenci objektů jazyka Python. Obdoba db4o pro platformy Java a .NET.

systémy, což umožní dokonalý přehled nad firemními procesy a lze tak optimalizovat a posunout informační technologie o úroveň výše. Mezi přednosti systému Zenoss patří:

- dohled nad aktivními prvky a sledování jejich vytížení.
- Možnost automatické rekonfigurace topologie v případě nečekaných událostí.
- Prevence, odhalování chyb a jejich rychlé řešení.
- Automatická detekce topologie a její následná vizualizace a mnohé další.



Obrázek 8 – topologická mapa Zenoss [19]

Pro všechny produkty diskutované výše je typické hned několik nevýhod:

- náročnost na konfiguraci a obslužný personál,
- u komerčních celků pak náklady na software samotný (licence),
- náklady na školení personálu.

Uvedené nevýhody plynou z charakteru takovýchto enerprise projektů snažících se pokrýt co největší možný rozsah služeb. Jejich snaha razantně převyšuje pouhou vizualizaci a jednoduchý pohled na síťovou topologii, navíc je nutné mít monitorovanou infrastrukturu plně pod kontrolou, protože v případě nedostupnosti SNMP na monitorovaných uzlech jen velice těžko bude schopen takový software pracovat ke spokojenosti svých uživatelů. [19], [20]

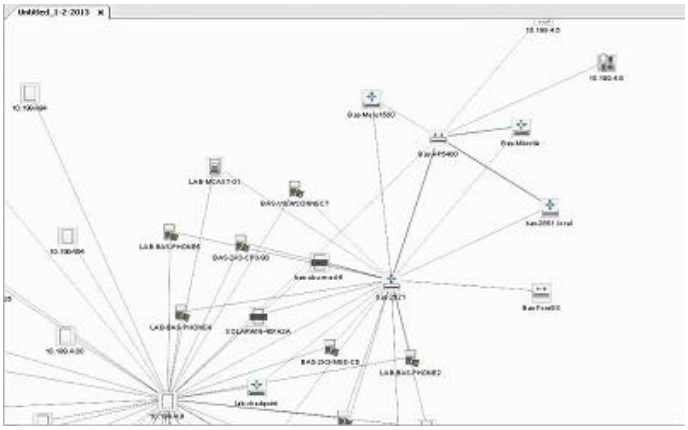
2.2 Topologie na úrovni středních sítí

Jedná se obvykle o aplikace nabízející pouze malou část z toho, co poskytují produkty diskutované v kapitole Topologie na úrovni velkých sítí. Na druhou stranu mají obvykle díky tomu rapidně nižší cenu, která se například u produktů spo1ečnosti SolarWinds pohybuje okolo 1 000 €, u 10-strikes pak 250 $ za licenci. Na rozdíl od komerčních výše popsaných produktu, kde je cena obvykle smluvní prostřednictvím kontaktu s obchodníkem a převyšuje mnohonásobně tyto částky.

2.2.1 SolarWinds Network Topology Mapper

Jde o komerční projekt určený pro střední třídu zabývající se mapováním a vizualizací topologie. Ke stažení je demoverze, která má dle výrobce omezení pouze na export map do formátu MS Visio. K dispozici je funkcionalita jako:

- automatická detekce topologie a její následná vizualizace.
- Reakce na změny v topologii.
- Podpora mnoha způsobů detekce SNMP, ICMP, WMI, CDP a další.
- Možnost exportu topologických map do formátů PDF, PNG a MS Visio.



Obrázek 9 – topologická mapa Network Topology Mapper [21]

Během testování demoverze se nepodařilo danou funkcionalitu ověřit. Byl testován rozsah sítě s prefixem 24 bitů (254 možných uzlů) a ve výsledku bylo nalezeno 35 uzlů, síť nebyla ve správě testera, takže nemohlo dojit k uplatnění možností SNMP protokolu. Skenování lze realizovat pouze pomocí průvodce a ten nenabízí podrobnější nastavení, jimiž by šel průběh ovlivnit. Po dokončení průvodce došlo sice k zobrazení nalezených uzlů, ovšem k vykreslení spojnic již nikoliv. Program působil poměrně neintuitivně a možnost k ručnímu dokreslení nebyla k nalezení. [22]

2.2.2 LanState

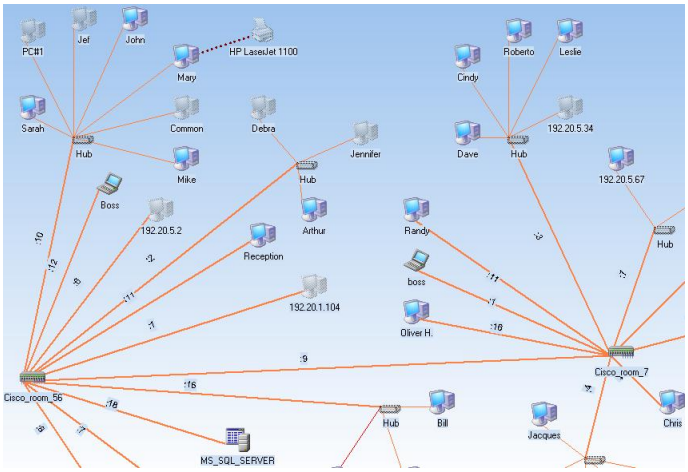
Komerční projekt společnosti 10-strike, který umožňuje generování topologických map s cílem urychlit přístup ke zdrojům na daných uzlech pomocí modelu. K dispozici je automatická detekce založená na bázi SNMP a vizualizace sítě.

V nabídce je zdarma ke stažení časově omezená verze. Produkt obsahuje několik modulů, jež doplňují jeho funkcionalitu, jako např.:

- modul pro automatickou detekci,
- modul pro sledování stavů daných uzlů,
- modul pro zasílání hlášení při výjimečných stavech. [23]

Při testování zkušební verze byla ověřena funkčnost detekce zadaných uzlů v rozsahu (testován stejný rozsah jako v předešlém případě), ale obdobně jako v předešlém případě chy-

běla vykreslená topologie. Nicméně program působil mnohem intuitivněji a nabídl možnost ručního propojení nalezených uzlů.



Obrázek 10 – topologická mapa LANState [24]

Do této kategorie spadají ještě další zástupci. Komplexní souhrn všech by ovšem přesahoval náplň této práce. Namátkou budou uvedeni jen někteří a to již bez doprovodných ilustrací topologických map, které lze většinou dohledat přímo na stránkách projektu:

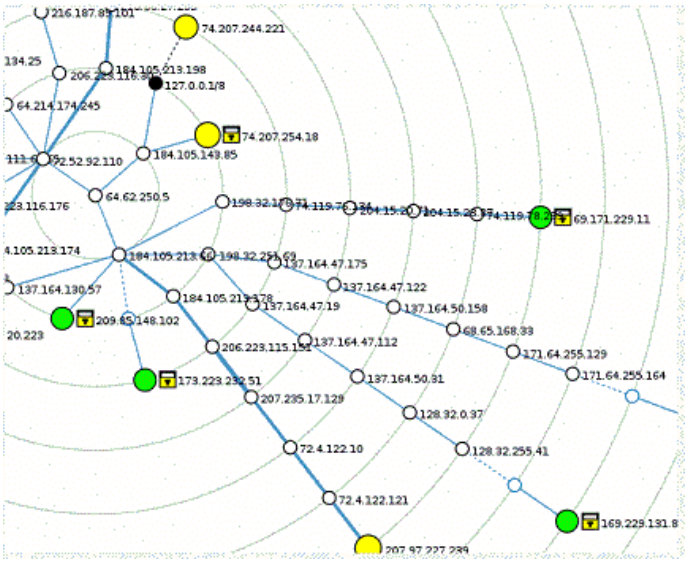
- OpManager,
- AccelOps,
- Icinga – OSS fork projektu Nagios,
- InterMappe,
- SNMPc8 Network Manager od firmy Castle Rock,
- WhatsUp Gold od společnosti Ipswitch, Inc.

2.3 Topologie na úrovni malých sítí

V této kategorii jsou obsaženy aplikace nebo spíše pomocné programy, které se soustředí primárně na detekci a zobrazení topologie a nenabízí žádné extra (nesoustředí se tolik na management) vlastnosti navíc, jako předešlé produkty. Víceméně se jedná o open sourcové záležitosti portované na jeden konkrétní operační systém a v případě komerčních záležitostí se cena za aplikaci pohybuje kolem pár dolarů.

2.3.1 Zenmap GUI

Grafická nadstavba známého síťového analyzátoru Nmap, která jej doplňuje o možnost vizuální reprezentace topologie analyzované sítě. Neumožňuje nad daným uzlem vykonat uživatelem definovaný příkaz. Aplikace využívá k zobrazení topologické mapy technologii flash. [25]

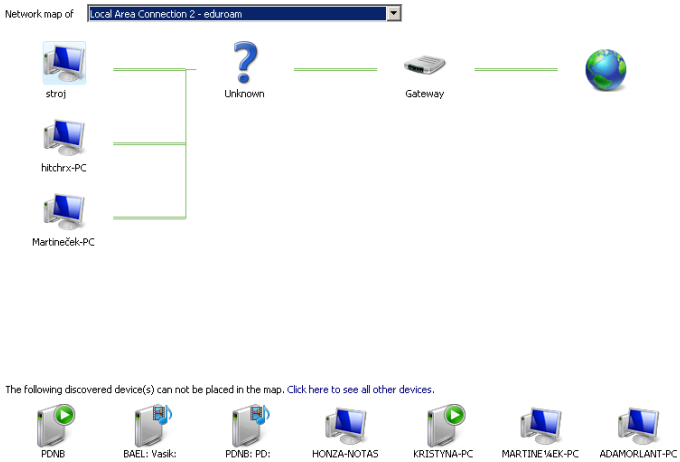


Obrázek 11 – topologická mapa Zenmap [26]

Při testování produktu došlo k vykreslení mapy, ovšem chyběly některé uzly na trase. U většího počtu skenovaných uzlů (řádově stovky) program vykazoval výrazné známky zpomalení. Nepodařilo se tedy jeho funkčnost více prozkoumat.

2.3.2 Link Layer Topology Discovery

Řada operačních systémů Windows nabízí od verze Windows 7 možnost zobrazit topologii lokální sítě pomocí protokolu LLDP, jež je dílem společnosti Microsoft. Tento nástroj má využití pouze v lokální síti[9]. [27]



Obrázek 12 – topologická mapa lokální sítě, LLTD MS Win. 7

2.3.3 NetworkView

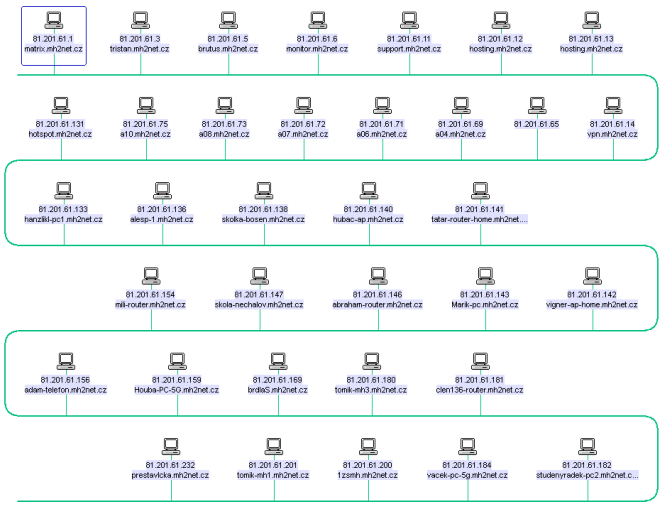
Jednoduchý program pro operační systémy z rodiny Windows. Nabízí detekci uzlů a jejich následnou grafickou reprezentaci. K dispozici je zdarma demo verze na 30 dní, jinak se za jednorázovou licenci platí zhruba do sto dolarů. V základní nabídce poskytuje sadu předdefinovaných příkazů, které je možné nad zvoleným uzlem vykonat. Mezi tyto patří napří-

[9] Testovací scéna pochází z lokální kolejní sítě Univerzita Pardubice, blok A.

klad telnet, ping, ftp klient, internetový prohlížeč a následně jsou pak ponechány 3 volné pozice pro uživatelem definované příkazy. Podporuje různé typy uzlů Server, Workstation, Unix station, Router, Pointer, … celkově 24 typů. Mezi jeho hlavní přednosti se řadí:

- Detekce uzlů a cest mezi nimi.
- Kontrolovat stav daných zařízení a v případě poruchy posílat varovná hlášení.
- Detekce výrobce daného hardware, dle jeho MAC adresy.
- K dispozici je vestavěný port scanner.
- Manuální editace a přidávání uzlů. [28]

V průběhu testování[10] zkušební verze byly ověřeny některé z funkcionalit, jako např. detekce topologie včetně její vizualizace a následné vyvolání příkazu nad daným uzlem. Výsledná topologická mapa ovšem neodpovídala skutečnosti, ani nebyla zobrazena varovná hláška upozorňující na tento fakt, jinými slovy daný model se jevil tak, že věrně kopíruje reálnou topologii, ale nebylo tomu tak.

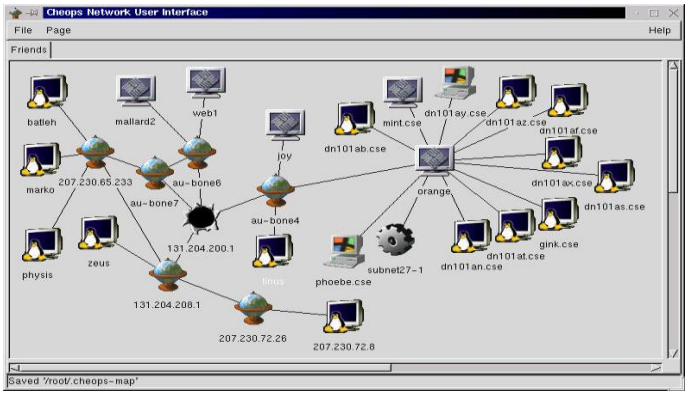


Obrázek 13 – topologická mapa NetworkView

2.3.4 Cheops-ng

Jednoduchý open sourcový program poskytující vizualizaci topologie včetně detekce operačního systému a služeb (dtto grafický skener). Vývoj byl v roce 2005 ukončen kvůli nedostatku času autora a malému zájmu nových vývojářů. [29]

[10] Opět byl použit stejný rozsah jako v předešlých kapitolách.

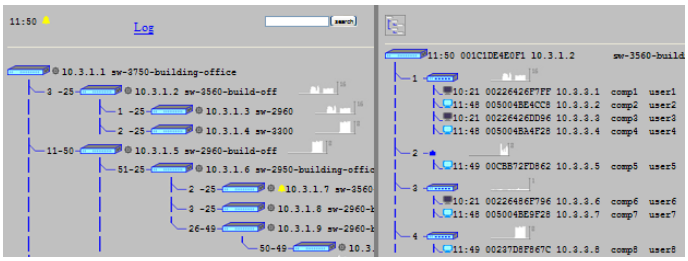


Obrázek 14 – topologická mapa Cheops [30]

2.3.5 Lantopolog

Poskytuje automatickou detekci topologie na úrovni lokální sítě a umožňuje ovládat aktivní prvky pomocí protokolu SNMP. K vyzkoušení je zdarma demo verze, jinak stojí licence na plnou verzi 30 $. Dále poskytuje:

- rychlý přehled o tom, do jakého portu je zapojeno jaké zařízení.
- Okamžitou reakci na změnu v topologii a propsání změny do grafické mapy.
- Monitorování stavu a v případě chyb posílat informační zprávy. [31]



Obrázek 15 – topologická mapa Lantopolog [31]

Při testování (na lokální síti) byl opět použit stejný rozsah jako v předešlých případech, ovšem detekováno bylo 0 uzlů, důvodem tomu je skutečnost absence SNMP na sledovaných prvcích. Jinou možnost k detekci program nenabízí, lze ho tedy doporučit pouze tam, kde má administrátor plně pod kontrolou dané prvky.

U programů z posledních dvou kategorii se podařila funkčnost ověřit pouze částečně nebo vůbec. K dispozici je obvykle verze pro jeden operační systém (až na výjimky). Některé aplikace jsou vázány na SNMP což znemožní jejich použití v sítích bez podpory tohoto protokolu. Jako nadějný kandidát se jevil z počátku program Zenmap. Nicméně nedovoloval definovat uživatelské příkazy a navíc vykazoval problémy s výkonem.

3 Způsoby detekce topologie

Vzhledem k orientaci práce na protokol IPv4 bude i rozbor technik respektovat toto zaměření. Novější varianta IPv6 nebude tedy podrobena analýze.

V následující pasáži budou probrány možnosti uplatnění protokolů od IP vrstvy výše, které je možné využit při detekci topologie počítačové sítě. Možnosti nižších vrstev nebudou uvažovány. Metodiky lze rozlišit na dvě větve.

3.1 Aktivní

Aktivním způsobem detekce se myslí proces, v němž figurují role klient/server a na výsledné podobě topologických dat se podílí oba účastníci (dva běžící procesy). Sledovaný uzel musí určitým způsobem poskytovat přístupové rozhraní, přes které bude komunikace probíhat a server musí zajistit korektní zpracováni dat.

Za nespornou výhodu těchto metod lze považovat prakticky neomezené možnosti. Je jen na tvůrci komunikačního protokolu, co zahrne do implementace. Například je možné z daného uzlu získat úplnou směrovací tabulku a topologii vybudovat na základě těchto údajů. Na druhou stranu je zde skutečnost, která znemožňuje využití tohoto postupu v sítích, které nemá iniciátor procesu detekce plně pod kontrolou.

Výhody:

- neomezené možnosti při získávání dat,
- lze vybudovat kompletní topologii odpovídající realitě.

Nevýhody:

- nutná podpora implementace protokolu na příslušném uzlu,
- zneužití a možná vyšší spotřeba systémových prostředků,
- předpoklad jisté počáteční znalosti[11] mapované sítě,
- u prvků pracujících čistě na síťové vrstvě znemožněno použití.

3.1.1 SNMP

Jedná se o protokol určený pro správu a dohled sítě, který je postaven na principech uvedených v odstavci výše. Rozlišuje dva účastníky komunikace: **agent** a **manažer**. Komunikace zpravidla probíhá ve směru manažer/agent. Manažer v intervalech zasílá požadavky a agent na ně patřičným způsobem reaguje. Ovšem na straně agenta lze definovat okolnosti, za kterých může komunikace probíhat opačným směrem. Agent v takovém případě kontaktuje manažera sám zasláním zprávy označované jako **trap**. Typicky je v dané síti spravované pomoci SNMP méně manažerů než agentů. V komunikačních zprávách figurují mimo jiné i takzvané identifikátory proměnných označované jako OID[12], které slouží jako

[11] Hesla, community string, případně IP adresy některých uzlů.

[12] Object identifier – jednoznačný identifikátor příslušné hodnoty v MIB stromu.

přístupový specifikátor ke zdrojům zařízení. Ty je možné číst, případně nastavovat. Seznam všech identifikátorů je uložen ve stromové databázi, která se označuje jako MIB[13]. Průchod od kořene k listu pak definuje konkrétní OID.

3.1.2 Komunikační rozhraní služeb

Do této kategorie spadají možnosti různých specifických démonů[14], kteří poskytují veřejné přístupové rozhraní. Prostřednictvím kterého lze programově získat data pro vybudování topologie. Namátkou to mohou být přístupové konzole k procesům implementujícím protokoly jako OSPF, RIP, BGP, SSH a další. [32]

3.1.3 Vlastní řešení

Poslední možnost, kterou lze považovat za aktivní způsob zjišťování informací, je vytvoření vlastního komunikačního protokolu nad TCP/IP, který bude zajišťovat sběr důležitých dat.

3.2 Pasivní

Za pasivní způsob detekce lze považovat situace, při nichž se využívá výhradně implementovaného chování protokolů TCP/IP. Pokud tedy zařízení daný protokol nebo jeho důležitou část podporuje, může být jeho chování využito při odhalování topologie. U této možnosti se podílí na detekci pouze jedna strana (jeden běžící proces), která vyhodnocuje reakce uzlů v síti na dané podněty a na základě nich pak utváří topologickou mapu. Tato kapitola pojednává o možnostech některých, z hlediska detekce zajímavých částí vybraných protokolů. Nemá suplovat zevrubný popis všech možných detailů.

Výhody:

- implementace na úrovní protokolu, tedy aplikačně nezávislé,
- žádná dodatečná režie na systémové prostředky,
- iniciátor detekce nemusí mít žádný administrativní přístup na uzly v mapované síti,
- není nutná prakticky[15] žádná znalost mapované sítě.

Nevýhody:

- topologická mapa nemusí věrně kopírovat originál, protože detekce odhalí pouze cestu vedoucí z jednoho místa k cíli, nikoliv však veškeré cesty mezi uzly.
- Možnost úmyslné blokace.

3.2.1 ICMP

Jedná se o servisní protokol nad IP, který se používá při oznamování událostí, jež nastaly v průběhu komunikace. Jeho primárním účelem není přenos dat mezi uzly jako je tomu u protokolů UDP nebo TCP, nýbrž přenos zpráv informačního charakteru.

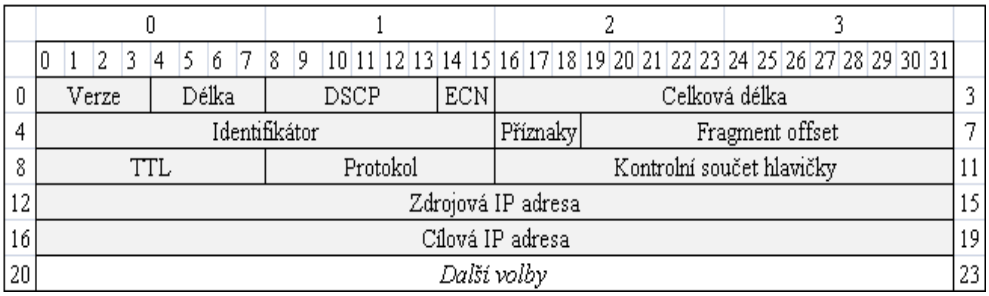
---

[13] Management information base – stromová struktura. Uchovávající veškeré proměnné reprezentující patřičné hodnoty sledovaných uzlů.

[14] Proces běžící na pozadí.

[15] Je nutné znát pouze rozsah mapované sítě.

Existuje mnoho typů zpráv, které lze pomocí ICMP signalizovat. Podrobněji však budou probrány pouze ty, jejichž chováni je možné využit při detekci topologie. Tyto zprávy jsou v podobě ICMP paketů řazeny přímo za IP záhlaví. [32]



Obrázek 16 – obecné IPv4 záhlaví

Stručný popis některých položek IP záhlaví:

- **Verze** – verze protokolu nabývá hodnoty 4 nebo 6 (záhlaví se liší mimo jiné i v jiných polích než je verze).
- **Identifikátor** – slouží k identifikaci fragmentů vzniklých při fragmentaci. Hodnota se využije v případě sestavování původního paketu z fragmentů a měla by být unikátní pro každou dvojici zdroj-cíl a protokol.
- **TTL** – maximální možný počet směrovačů, přes které může paket projít. Směrovač je povinen toto pole snižovat minimálně o hodnotu jedna, aby tak zabránil nekonečnému putování paketu sítí.
- **Protokol** – číselná identifikace protokolu, který následuje za IP záhlavím, např.
  - 1 značí ICMP,
  - 6 značí TCP,
  - 17 značí UDP a jiné.
- **Zdrojová a cílová adresa** – 32 bitové identifikátory zdrojové a odesílající strany.
- **Další položky** – položky z obrázku výše jsou definovány jako povinné (20 B), implementace protokolu však dává možnost využít dalších voleb o celkové velikosti 40B (maximální velikost IP záhlaví totiž činí 60 B).

Zajímavým doplňkem z hlediska odhalování topologie je volba **zaznamenávej směrovače**, která signalizuje směrovačům, přes něž paket putuje, aby zaznamenaly adresu svého výstupního rozhraní do seznamu v IP záhlaví. Celkem je možné uchovat 9 adres, což pramení z omezené 40 B velikosti pro doplňkové položky. Pokud tuto volbu podporuje i dotazovaná strana, je možné získat adresy obou rozhraní směrovače, přes které paket putoval (za předpokladu, že cíl zkopíruje doručené adresy).

Byť se tato volba, jeví, jako použitelný způsob, má několik nedostatků:

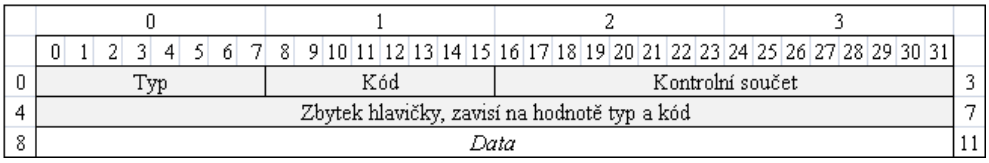
- omezení na max. 9 záznamů,
- z bezpečnostních důvodů nebývá implementována, případně je administrativně blokována. Tento nedostatek byl v rámci fáze testování různých způsobů detekce po-

tvrzen autorem této práce, čímž lze tuto volbu doporučit pouze pro experimentální účely.

Podrobnější popis určitých polí v IP záhlaví bude rozebrán z hlediska programového zpracování až v praktické části. V této kapitole je zmínka uvedena pouze z důvodů:

- získání představy o existenci těchto polí,
- souvislosti mezi přenášeným obsahem a IP záhlavím.

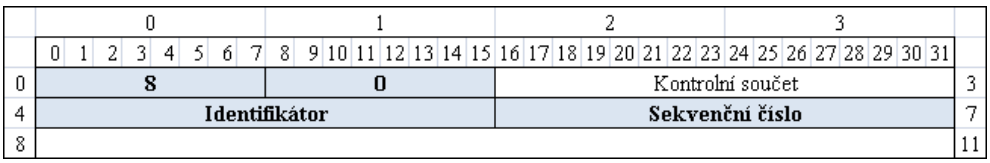
Dále se kapitola věnuje čistě množině zpráv z rodiny ICMP. Obecné záhlaví má formát shodný s obrázkem níže.



Obrázek 17 – obecné ICMP záhlaví

- **Typ** – označuje druh neseného paketu. Dále v textu budou detailněji rozebrány typy 0, 3, 8 a 11.
- **Kód** – slouží k další kategorizaci pole typ.
- **Kontrolní součet** – je suma přes všechny pole záhlaví a dat, kde pole kontrolní součet má v době kalkulace hodnotu 0.
- **Zbytkové pole záhlaví** – je závislé na polích typ a kód.

Typ 8 (echo reply) – žádost o odpověď se používá v případě, kdy je nutné ověřit, zda je některý uzel v síti dostupný. Zpráva se odešle na cílový uzel a čeká se, zda stanice odešle patřičnou odpověď zpět. Jako typ je nastavena hodnota 8 a kód 0. Zbytková část má pak význam identifikátoru a sekvenčního čísla. Tato pole mají smysl v případech, kdy je nutné spárovat žádost o odpověď s odpovědí.



Obrázek 18 – ICMP typ 8

Typ 0 (reply) – jde o odpověď na předešlý typ 8. Záhlaví se liší pouze v poli typ, to obsahuje hodnotu 0, kód a význam zbytku polí zůstává totožný.

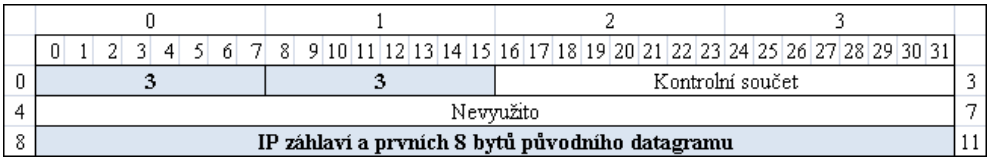
Princip těchto zpráv je možné ze své podstaty využít k detekci existence uzlů v síti.

Typ 3 (destination unreachable) – tato zpráva se odesílá v případech, kdy je nutné odesílající stranu upozornit na určitou skutečnost jako např. nedostupná služba nebo síť. Pole kód může nabývat hodnot 0 až 15.

Podrobněji je rozebrána zpráva s kódem 3 (nedostupný port), kterou generuje uzel v případě, že nemůže přijmout spojení na definovaném portu. V datové části se vrací:

- původní IP záhlaví (typicky 20 B),
- prvních 8 B původního neseného paketu (nejčastěji UDP nebo TCP záhlaví).

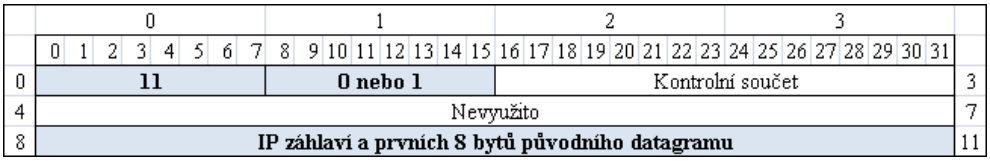
Tato skutečnost je důležitá v případě identifikace paketu odesílající stranou, které byl navrácen (jeho část), protože nemohl být doručen.



Obrázek 19 – ICMP typ 3

Tuto zprávu lze využít k odhalení uzlu v síti v případech, kdy se úmyslně realizuje spojení na neexistující port, načež daný uzel odpoví (pokud není chování explicitně potlačeno) právě zmíněnou ICMP zprávou typu 3 s kódem 3. Analogické chování nabízí tatáž zpráva s kódem 2, kterou odpoví uzel nepodporující daný transportní protokol.

Typ 11 (time exceeded) – zpráva s kódem **0** informuje o skutečnosti, že vypršela platnost paketu putujícího sítí. Každý paket je opatřen v IP záhlaví hodnotou TTL definující počet přeskoků, přes které může projít. Při průchodu je každý směrovač povinen toto pole snížit minimálně o jedna, aby tak zabránil nekonečnému putování paketu sítí. V případě, že je kód nastaven na hodnotu **1**, jde o zprávu indikující stav, při kterém není možné z fragmentů v čas sestavit celý paket, např. když se nějaké fragmenty cestou ztratí, poškodí. Zbytkové pole má stejný význam jako u výše popsaného typu 3 a slouží k identifikaci navráceného paketu.



Obrázek 20 – ICMP typ 11

Zprávu tohoto druhu lze využít při zjišťování existence směrovačů na trase vedoucí od zdroje k definovanému cíli. Vychází se z principu, který implementuje nástroj traceroute. Tedy, že každý směrovač snižuje hodnotu TTL a pokud ta dosáhla hodnoty 0, je vygenerována ICMP zpráva typu 11. Postupně jsou odesílány pakety s hodnotami 1, 2, 3, … První směrovač hodnotu 1 sníží na 0 a generuje zprávu o vypršení platnosti, druhý směrovač sníží hodnotu 2 na 0 atd. Na principech tohoto chování lze sestavit trasu putování paketů sítí.

V situacích, kdy chce správce sítě skrýt topologii, může učinit opatření, které toto standardní chování potlačí, a tak znemožnit detekci. Obecně to však není doporučná technika, protože může dojít k vytváření cyklů.

3.2.2 UDP

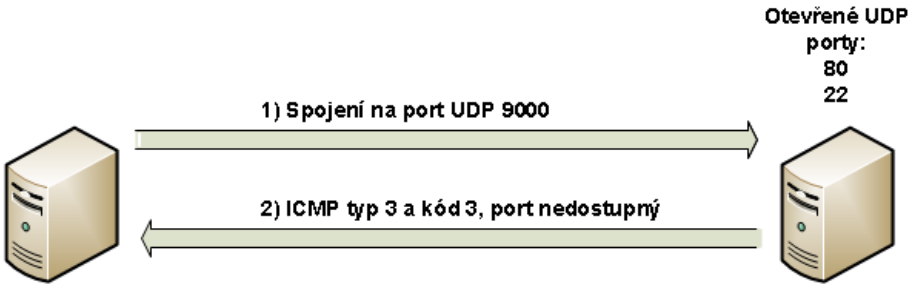
Jde o protokol používaný pro přenos dat mezi klientskými aplikacemi. Svým určením spadá do transportní vrstvy. Nenabízí při přenosu dat žádnou garanci na doručení, tu musí v případě potřeby implementovat přímo programátor aplikace. Obsahuje jednoduché záhlaví, které se skládá z následujících polí [32]:

- **zdrojový a cílový port** – oba slouží jako komunikační kanál mezi aplikacemi,
- **délka** – zahrnuje záhlaví (8B) a data,
- **kontrolní součet** – kalkulovaný přes záhlaví a data.

	0								1								2								3								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0	**Zdrojový port**																**Cílový port**																3
4	Délka																Kontrolní součet																7
8	**Datová část**																																11

Obrázek 21 – UDP záhlaví

Určité chování vyvolané v souvislosti s tímto protokolem je možné využít způsobem, který byl naznačen v popisu ICMP zprávy signalizující nedostupnou službu. V případě realizace spojení na neexistující port, je cílový uzel podnícen k informování protistrany o nedostupnosti portu. Pokud se ovšem spojení zdaří, je nutné ošetřit stav na základě konkrétní aplikace naslouchající na daném portu.



Obrázek 22 – princip detekce pomocí UDP

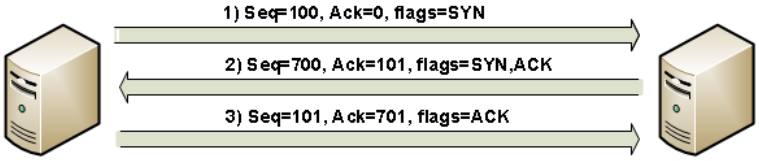
3.2.3 TCP

Jedná se o protokol jistým způsobem podobný, co se týče účelu, jako UDP. Rozdíl je v tom, že TCP poskytuje mechanismus pro garanci doručení dat pomocí třícestného navazování spojení. [32]

	0								1								2								3								
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
0	Zdrojový port																Cílový port																3
4	Sekvenční číslo																																7
8	Potvrzovací číslo																																11
12	Velikost				Rezerva				Příznaky								Velikost okna																15
16	Kontrolní součet																Urgent																19
20	*Další volby*																																23

Obrázek 23 – obecné TCP záhlaví

1. Strana A provede mimo jiné nastavení polí:
    - **potvrzovací číslo** – na hodnotu 0,
    - **číslo sekvence** – náhodné číslo Ax,
    - **příznaky** – na hodnotu SYN a posléze odesílá straně B.
2. Strana B provede v návaznosti na krok 1 nastavení polí:
    - **potvrzovací číslo** – na Ax+1,
    - **číslo sekvence** – náhodné číslo By
    - **příznaky** – na hodnoty SYN, ACK a odesílá zpět straně A.
3. Strana A provede nastavení polí:
    - **potvrzovací číslo** – na By+1,
    - **číslo sekvence** – na Ax+1,
    - **příznaky** – na hodnotu ACK a odesílá zpět straně B [32].



Obrázek 24 – třícestný TCP handshake

I u tohoto protokolu je možné si povšimnout jistých implementačních rysů, které lze použit ve prospěch odhalování aktivních uzlů. Ty rozdělují možnosti na několik případů:

- navázání spojení na neexistující port, jak bylo popsáno výše v odstavci UDP.
- Využití mechanismu navazování spojení, pokud proběhne úspěšné přijetí datagramu z fáze dvě, lze považovat daný uzel za aktivní.
- Pokud se podaří spojení navázat, bylo by nutné ošetřit chování na aplikační úrovni. Nicméně existenci uzlu lze dokázat již z fáze dvě, třícestného navazování.

4 Možnosti a úskalí vizualizace grafové struktury

Tento odstavec pojednává o problematice zobrazení grafové struktury ve 2D prostoru. Dále se zaobírá problémy, které tato doména obsahuje a jejich možným řešením.

4.1 Grafová struktura

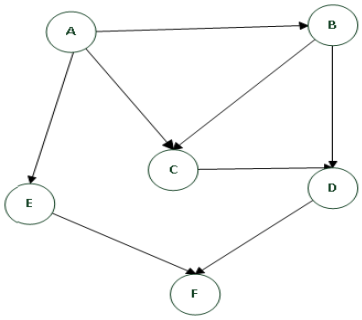
V této pasáži budou popsány základy problematiky týkající se grafů pro potřeby modelování topologie počítačové sítě. Zevrubný popis není předmětem této práce, tím se zabývá samostatná vědní disciplína teorie grafů.

Graf slouží jako zjednodušený model pro popis objektů reálného světa a lze pomocí něj vyjádřit např. topologii dopravní, počítačové nebo sociální sítě. Je tvořen množinou elementů, mezi nimiž jsou vyjádřeny určitým způsobem vztahy.

Elementy jsou dvojího druhu:

- vrcholy,
- hrany.

Vrcholy představují základní stavební bloky grafu a hrany reprezentují vztahy mezi vrcholy. Na základě definice vztahu lze graf rozlišit na **orientovaný**, **neorientovaný** nebo **kombinovaný**.



Obrázek 25 – orientovaný graf

Pod pojem **orientovaný graf** je chápána struktura, ve které mají hrany definovánu svou jednosměrnou orientaci. U **neorientovaného grafu** se předpokládá, že jsou hrany obousměrné. **Kombinovaný graf** pak obsahuje hrany obou typů.

Graf sám o sobě definuje pouze vrcholy, hrany, čímž vyjadřuje síť vztahů. Již ale nedefinuje žádné dodatečné informace potřebné k zobrazení dat v euklidovském prostoru. Jinými slovy nejsou k dispozici souřadnice pro zobrazení ve 2D prostoru. S vizualizací hierarchických struktur souvisí několik problémů:

- absence koordinátů x,y, které definují polohu vrcholu ve 2D prostoru.
- Rozložení vrcholů v prostoru tak, aby působilo dostatečně esteticky.

Problémy týkající se rozložení dat v prostoru řeší vědní disciplíny z oblastí geometrie, teorie grafů a počítačové zobrazení informací.

4.2 Algoritmy rozložení

Vstupem těchto algoritmů je grafová struktura s vrcholy bez přiřazených souřadnic a na výstupu je každému z vrcholů přiřazen 2D koordinát. Jejich cílem je vytvořit esteticky vhodné zobrazení. Algoritmů a jejich implementací je celá řada, stručně budou popsány obecné postupy a principy fungování a pro lepší představu doplněny ilustrace některých implementací.

4.2.1 Algoritmy řízené silou

Jedná se o metody, které určují polohu vrcholu v ploše na principu simulace fyzikálního procesu jako je např. mechanika molekul, systémy pružin, působení elektrostatické nebo gravitační síly a případně jiné.

Mezi vrcholy tedy dochází k působení jistých druhů sil, za pomocí nichž se algoritmicky formuje finální pozice vrcholu. Cílem je dosáhnout stav, který bude odpovídat situaci, při které jsou vrcholy od sebe odděleny co nejlépe a hrany mají co možná nejkratší možnou délku.

Typicky se pro vzájemné přiblížení dvou vrcholů využívá principu pružin (hrana je nahrazena pružinou dle definic Hookova zákona) a k oddělení vrcholů se využívá systému elektricky nabitých částic. Jde o iterativní algoritmy, které v každé iteraci simulují fyzikální proces působení sil do té doby, než systém přejde do rovnovážného stavu, tj. relativní poloha vrcholů se mezi iteracemi již nemění. V případě, že se systém nachází v rovnováze, jsou pozice vrcholů označeny za definitivní a může dojit k vykreslení grafu. [33]

Výhody:

- relativně rozumné výsledky pro grafy menších velikostí (50 - 100 vrcholů),
- flexibilita, tedy snadná možnost rozšíření např. pro 3D prostor.
- Jednoduché na pochopení, protože jsou založeny na všeobecně známých fyzikálních principech a lze tedy predikovat jejich chování,
- jednoduché na implementaci.

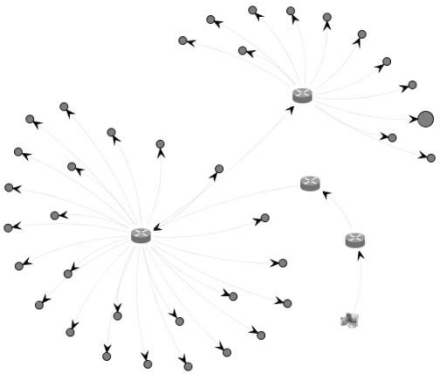
Nevýhody:

- doba chodu, tyto algoritmy mají poměrně vysokou složitost $O(n^3)$, kde n je počet vrcholů daného grafu. Tato složitost pramení ze skutečnosti, že je nutné v každé iteraci vypočítat působení sil mezi všemi páry vrcholů. Ovšem toto lze optimalizovat, protože přitažlivé síly mají lokální charakter a problém vypočtu, je možné rozdělit pouze na sousední vrcholy a zmenšit tak celkový počet kombinací.

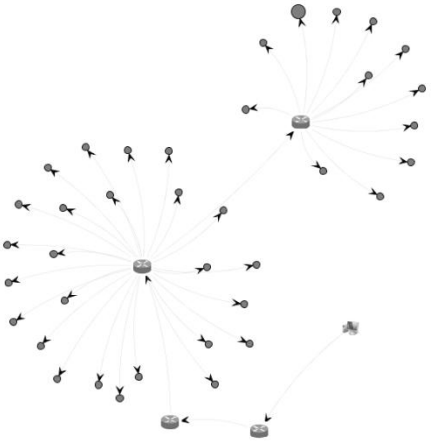
Mezi známé a používané implementace tohoto typu patří:

- Fruchterman-Reingold,

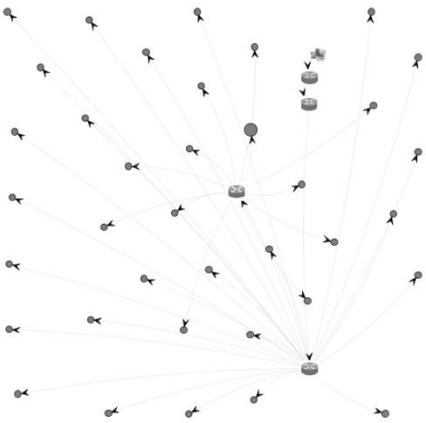
- Kamada-Kawai,
- Spring.



Obrázek 26 – ukázka rozložení vrcholů Kamada-Kawau



Obrázek 27 – ukázka rozložení vrcholů Frcuhterman-Reinghold

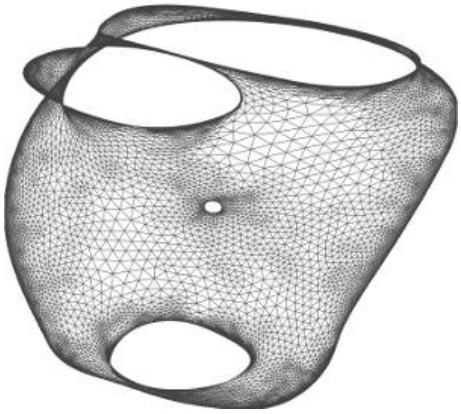


Obrázek 28 – ukázka rozložení vrcholů Spring

Pomocí těchto algoritmů je možné vytvořit relativně slušný model odrážející skutečnou topologii počítačové sítě, viz implementace Fruchterman-Reinghold nebo Kamada-Kawai. Téměř všechny důvody pro jejich použití jsou shrnuty v odstavci výhody, výše.

4.2.2 Spektrální algoritmy

Metody vycházející ze spektrální teorie grafů, při které se zkoumají různé vlastnosti grafu ve vztahu k charakteristickému polynomu[16], vlastním číslům a vlastním vektorům vycházejících z matic jako je Laplaceova (využívá se pro výpočet kostry[17] grafu) nebo matice sousednosti. Algoritmy pak používají vlastní vektory matice jako souřadnice pro vrcholy. [34]



Obrázek 29 – výstup spektrálního algoritmu [35]

Pro vizualizaci topologie počítačové sítě nelze doporučit vzhledem ke své složitosti odrážející řešení NP problémů i přesto, že některé z nich je možné řešit numericky. Jsou vhodné spíše pro sofistikovanější tvary, než je topologie počítačové sítě, viz Obrázek 29 – výstup spektrálního algoritmu Obrázek 29 – výstup spektrálního algoritmu

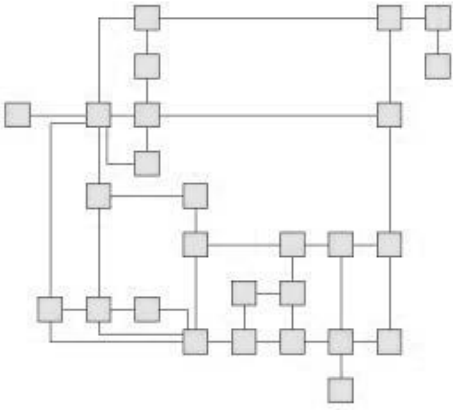
4.2.3 Ortogonální algoritmy

Někdy též algoritmy pravoúhlého zobrazení spočívají v kreslení hran horizontálním nebo vertikálním způsobem tak, aby byly kolmé s osami souřadného systému. Původně tyto metody řešily problémy rozložení součástí u tvorby integrovaných obvodů a při navrhování plošných spojů. Následně pak byly účelně modifikovány pro tvorbu rozložení vrcholů ve 2D ploše.

Skládají se z více fází, ve kterých se snaží vstupní graf transformovat do rovinného, odstraněním průniků křížících se hran. Orientace hran je volena tak, aby došlo k minimalizaci ohybů. Vrcholy jsou pak umístěny shodně s touto orientací, zároveň dochází i k minimalizaci velikosti plochy potřebné ke kreslení. [34]

[16] Polynom, jehož kořeny představují vlastní čísla dané matice.

[17] Kostra grafu je libovolný podgraf, který hranami spojuje všechny vrcholy původního grafu a zároveň sám neobsahuje žádnou kružnici, jinými slovy se jedná o strom.

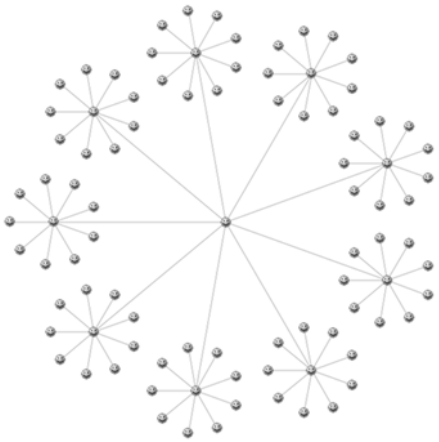


Obrázek 30 – výstup ortogonální metody rozložení [36]

Tento algoritmus je vhodný především pro programově orientované kreslení modelů a schémat, protože ne vždy je možné pomocí pravoúhlých cest dostatečně přesně vystihnout patřičné detaily počítačové sítě. Případně by její zobrazení působilo nevzhledně a navíc ne vždy je možné dosáhnout planarity grafu.

4.2.4 Stromově orientované algoritmy

Algoritmus vhodný pro stromově orientované struktury, při kterém se přímí potomci uzlu kreslí na kružnici okolo a svým způsobem jej obalují. Nižší hierarchické úrovně mají menší poloměr kruhu, takže nedochází ke křížení, v anglické literatuře bývá též označován jako **balloon layout**, případně **tree like layout**. [34]

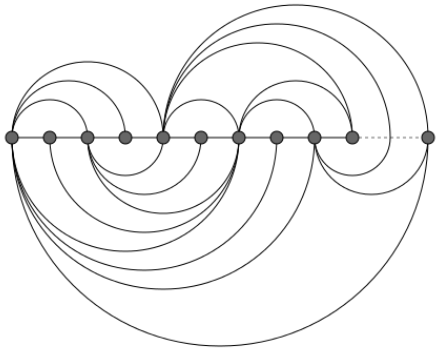


Obrázek 31 – výstup stromově orientovaného algoritmu [37]

Tento algoritmus nabízí relativně slušné estetické zobrazení, ale jeho vyžití pro popis topologické mapy počítačové sítě lze doporučit pouze v případě, že topologii je možné popsat stromovou strukturou, tedy že vrchol má vždy pouze jednoho předka. Což u modelu počítačové sítě nemusí být vždy pravda. Několik cest může směřovat přes jeden uzel viz Obrázek 25 – orientovaný graf, např. vrchol C. Svým způsobem, jde o jistý druh rovinného grafu. Algoritmus orientován výhradně na stromy, proto jeho využití při kreslení obecného grafu nelze doporučit.

4.2.5 Algoritmus lineárního vkládání

Vrcholy se umístí na přímku a hrany jsou kresleny pomocí oblouků nad nebo pod přímkou [34].

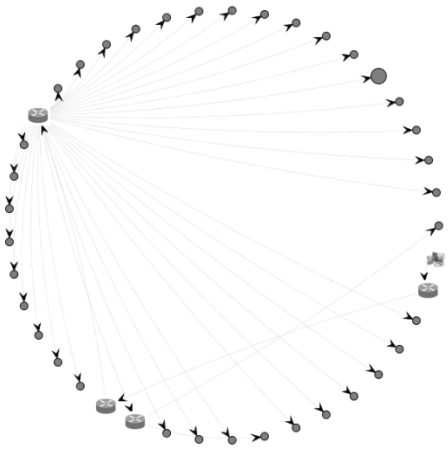


Obrázek 32 – výstup algoritmu lineárního vkládání [38]

Tento algoritmus není díky svému charakteru vhodný pro popis topologické mapy.

4.2.6 Kruhový algoritmus

Jednoduchá metoda kreslení grafu, při které se vrcholy uspořádají rovnoměrně do kruhu a vytvoří tak pravidelný mnohoúhelník. Všechny vrcholy mají pak od středu kruhu stejnou vzdálenost. [34]



Obrázek 33 – výstup algoritmu pro kruhové rozložení

V tomto konkrétním případě je vykresleno cca 40 vrcholů, rozložení působí poměrně esteticky, nicméně daná mapa neodpovídá reálné topologii, protože vrcholy jsou všechny zobrazeny na stejné úrovni. Nerozlišují se tedy uzly, přes které vedou další cesty od těch koncových.

Tento algoritmus lze doporučit u menších sítí s jedním centrálním prvkem, který lze po vykreslení posunout do středu. Případně jej začlenit v kombinaci s jinou metodou rozložení.

4.2.7 Metoda vrstveného kreslení

Známá též jako Sugiyamův[18] styl je vhodná pro acyklické orientované grafy nebo grafy, které se jim přibližují. Algoritmus pracuje v několika krocích, nejprve se vrcholy rozmístí horizontálně po vrstvách (řádcích) a hrany míří směrem dolů z jedné vrstvy do druhé. V dalším kroku se vrcholy uspořádají v každé vrstvě tak, aby se eliminovalo křížení hran. Stručný popis jednotlivých kroku lze shrnout v několika bodech.

1. Pokud se nejedná o acyklický orientovaný graf (AOG), dojde k nalezení množiny hran, které jej činí cyklickým a k jejich otočení tak, aby došlo k likvidaci cyklu. Ovšem nalézt množinu, jež obsahuje minimálně jednu hranu každého cyklu v grafu, je problém označovaný v anglické literatuře jako **NP-complete feedback arc set**, který se řeší pomocí algoritmu hladových heuristik[19], protože pro jeho řešení neexistuje žádný efektivní způsob.

2. Výstupem předešlého kroku je tedy AOG, přičemž jeho vrcholy jsou následně přiřazeny do vrstev (snaha je produkovat spíše méně vrstev než více), tak aby hrany směřovaly z vyšší vrstvy do nižší a zároveň bylo dosaženo rovnoměrného rozmístění vrcholů. Při řešení tohoto kroku se vychází z Mirského teorému, který říká, že přiřazení vrcholů do vrstev je závislé na délce nejdelší cesty[20]. Na rozdíl od problému hledání nejkratší cesty, který lze vyřešit za jistých předpokladů[21] v polynomiálním čase, se hledání nejdelší cesty řadí do skupiny NP-úplných. K řešení rozvrstvení lze využít Coffman-Grahamův algoritmus nebo využít metod lineárního programování.

3. Hrany, které překrývají více vrstev, jsou nahrazeny cestou mezi fiktivními vrcholy tak, aby v rozšířeném grafu každá hrana propojovala pouze dva vrcholy v sousedních vrstvách.

4. Vrcholy v jednotlivých vrstvách jsou přeuspořádány tak, aby se eliminovalo křížení hran mezi sousedními vrstvami. Ovšem nalézt minimální nebo maximální počet křížení je opět NP-úplný problém. Takže se k řešení využívají heuristiky jako např. určení pozice vrcholu jako průměr nebo medián z pozice sousedních vrcholů z předešlé úrovně a následné prohazování sousedních párů tak, aby se snížil počet křížení.

5. Každému vrcholu je přiřazena odpovídající souřadnice v souladu s předchozím krokem.

6. V tomto kroku dojde ke zpětnému převrácení hran z prvního kroku a k odstranění fiktivních vrcholů, načež může dojít k samotnému vykreslení grafu. Aby se zabrá-

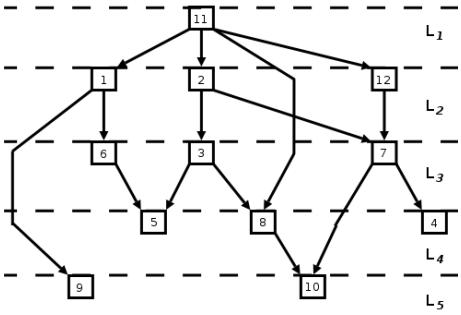
[18] Kozo Sugiyama – autor, který poprvé představil tento způsob kreslení grafu.

[19] Hladový algoritmus je jedním z možných způsobů řešení optimalizačních úloh. V každém svém kroku vybírá lokální minimum, přičemž existuje šance, že takto nalezne minimum globální.

[20] Snaha nalézt nejdelší možnou cestu mezi vrcholy, tak aby se žádný vrchol v cestě neopakoval.

[21] Graf, jež nemá záporně ohodnocené hrany a cykly.

nilo průnikům mezi jednotlivými vrcholy a hranami, mohou být hrany, které vedou přes více vrstev, zobrazeny jako spline křivky vedoucí přes pozice fiktivních vrcholů. [39]

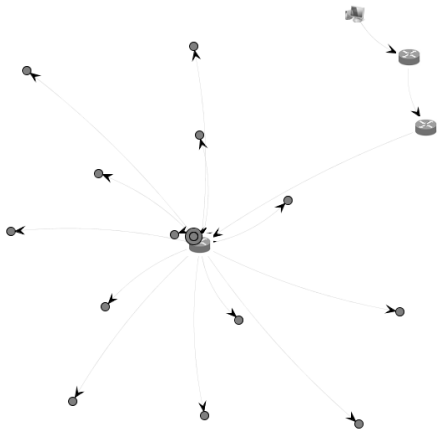


Obrázek 34 – výstup algoritmu pro vrstvené rozložení [40]

Algoritmus je vzhledem k výskytu problémů z rodiny NP-úplných nevhodný kvůli své složitosti.

4.2.8 Ostatní algoritmy

Do této kategorie mohou spadat např. metody založené na teorii neuronových sítí, které jsou postaveny na základech samo-organizujících map (Kohonenovy mapy), které shlukují objekty se společnými vlastnostmi (u grafu se jedná tedy o vrcholy). Mezi známé implementace tohoto druhu patří např. **Meyer's self-organizing graph layout**. [41]



Obrázek 35 – Meyerův algoritmus rozložení

Pro popis topologie je tento algoritmus zcela nevhodný vzhledem k tomu, že vytváří shluky vrcholů, které se překrývají a tím znemožňují čitelnost modelu.

5 Návrh a realizace vlastního řešení

Jak již bylo zmíněno v úvodu práce, praktická část je zaměřena na vytvoření aplikace, která bude mapovat a vhodným způsobem zobrazovat topologickou mapu počítačové sítě.

K mapování topologie byl zvolen **pasivní způsob**, jehož výhody byly diskutovány v odstavci Pasivní. Zejména však proto, že tento způsob detekce umožní aplikaci využít i v sítích, které nemá uživatel ve správě a navíc její výstup může posloužit jako vhodný prostředek pro počáteční orientaci v neznámém prostředí.

Pro vývoj aplikace byla zvolena platforma Java, jež nabízí stabilní základ pro tvorbu grafických více vláknových aplikací a výsledný produkt je tak použitelný nezávisle na operačním systému. Technologie a knihovny, které jsou použity při tvorbě, budou detailně popsány v následujících odstavcích.

5.1 Požadavky na aplikaci

Primární požadavek je ten, že aplikace umožní automatickou tvorbu modelu reálné počítačové sítě. Na vstupu bude uživatelem zadán rozsah síťových adres a na výstupu se zobrazí model počítačové sítě. Zmíněná funkcionalita se skládá ze čtyř částí:

1. detekce aktivních uzlů v rozsahu,
2. detekce cest mezi těmito uzly,
3. vizualizace dat získaných z předešlých dvou kroků,
4. uživatelská interakce.

Na bod čtyři jsou kladena ještě další omezující kritéria a to zejména možnost manipulace s jedním, případně skupinou uzlů tak, aby měl uživatel možnost vlastnoručního rozložení, které se při automatické detekci řídí algoritmicky. Toto rozšíření umožní výsledný model co nejvíce přizpůsobit reálnému.

Dále aplikace umožní zvýraznit směrovače, přes které vede trasa. Tedy z počátečního (ten, ze kterého probíhá detekce) do cílového uzlu (uživatelem zvolený).

Možnosti interakce rozšiřuje i vlastnost definovat externí příkazy, které je možné pro vybraný uzel vykonat, jako např. vyvolat přihlašovací terminál, internetový prohlížeč, nástroje typu traceroute, ping a další. Pomocí externího programu traceroute může uživatel ověřit, zda vykreslená trasa odpovídá reálné, ta se může pomocí protokolů pro dynamické směrování totiž měnit. Aplikace je vůči této skutečnosti imunní.

Pro podporu rychlého rozhodování při výskytu nečekaných potíží (výpadek uzlu) aplikace implementuje mechanismus, který zajistí vhodnou signalizaci těchto jevů.

Omezení týkající se nasazení aplikace jsou:

- na sítě malého až středního rozsahu. Řádově by měla být aplikace schopna bez větších obtíží zpracovat desítky až stovky uzlů.

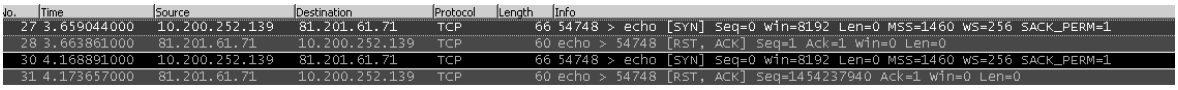
- Orientace pouze na sítě se statickým směrováním, tedy že aplikace nebude reagovat na změny v topologii.

5.2 Knihovny pro práci se sítí

Jak již bylo zmíněno v odstavci Návrh a realizace vlastního řešení, jako základ pro vývoj byl zvolen jazyk Java, což omezuje i výběr knihoven potřebných pro vývoj na této platformě.

První co bylo nutné při implementaci vyřešit, byl výběr vhodné knihovny, která umožní úplný přístup k síťovému RAW socketu operačního systému. Tento přístup se může systém od systému lišit. Proto tvůrci jazyka nezahrnuli tuto podporu přímo do JDK a je nutné přilinkovat potřebné knihovny, které tuto podporu přidávají.

Java sice nabízí jistou podporu pro síť v balíčku *java.net*, ovšem ne na takové úrovni, jaká byla potřebná v této práci. Při vývoji bylo nutné mít vytváření a inspekci paketů plně pod kontrolou. Např. Java umožňuje ověřit pomocí vestavěného API dostupnost uzlu pouze na úrovni TCP tím, že se pokusí připojit na port 7 (echo) a pokud se TCP handshake podaří, vyhodnotí se dostupnost uzlu jako pravdivá.



Obrázek 36 – pokus o ověření dostupnosti hosta pomoci Java API

Přirozeně se zde nabízí možnost vytvořit pomocí třídy *Runtime* nový proces externího příkazu, např. ping a jeho výstup transformovat do přijatelné podoby. Ovšem toto řešení je poněkud těžkopádné a vzniká jistá závislost na nástrojích operačního systému. Navíc je zde omezeno použití výhod vláken pro urychlení práce, protože vytvořit 100 procesů je mnohem méně efektivní, než vytvořit a řídit stejný počet vláken.

K dispozici je několik knihoven umožňujících požadovanou funkcionalitu.

5.2.1 Jpcap – sourceforge verze

Pomocí této knihovny je možné pakety zachytávat, analyzovat. Nadále je zde i určitá podpora vizualizace. Nicméně tato knihovna neumí pakety odesílat a její vývoj byl roku 2004 ukončen. Produkt je ale stále dostupný na serveru *sourceforge.net*. Pro použití v projektu je tato knihovna nevhodná kvůli absenci podpory pro odesílání paketů [42].

5.2.2 Jpcap – Keita Fuji verze

S předešlou knihovnou má společné pouze jméno, jinak se jedná o úplně odlišnou implementaci, kterou vytvořil Keita Fuji za svou dobu působení výzkumného asistenta na University of California, Irvine a licencoval ji pod LGPL. S koncem svého působení na univerzitě roku 2007 skončila bohužel i podpora a další vývoj této knihovny. Poslední dostupná verze je označena jako 0.7, která je stále k dispozici. Návrh je však na relativně dobré úrovní a díky otevřenému kódu a licenci je možné ji nadále rozšiřovat dle příslušných potřeb. V průběhu psaní (únor 2013) praktické části byl projekt dostupný na webu univerzity,

kde autor působil. Nyní (duben 2013) je dostupná pouze zprostředkovaně na mirrorech nadšenců.

Jde o jakýsi most mezi API jazyka C a Javou. Využívá knihoven libpcap/winpcap, takže její použití je možné všude tam, kde je jejich podpora. Přemostění je zajištěno pomocí JNI[22]. Umožňuje pracovat s protokoly jako IPv4, IPv6, ARP/RARP, TCP, UDP a ICMPv4. Pro každý typ protokolu je vytvořen odpovídající Java objekt, který je možné v prostředí Javy pohodlně používat. Pakety zachycené pomocí Jpcap jsou nadále směřovány ke svému původnímu cíly, není tedy možné měnit jejich směr (pracuje se s kopií). Souhrn charakteristických vlastností je zhruba následující:

- jednoduché API a dokumentace,
- dostupné vzorové příklady,
- podpora zachytávání i odesílání paketů,
- podpora IPv4 a IPv6.

Pro vývoj byla tedy zvolena tato verze knihovny.

V průběhu práce s knihovnou byly zjištěny jisté fatální nedostatky a to zejména při práci s ICMP pakety, kdy pro pole **sekvenční číslo** a **identifikátor** byl alokován (chyba ve špatném bitovém posunu) pouze jeden byte namísto dvou, což omezovalo platnost pro hodnoty těchto polí na max. 127 (Java nabízí pro byte rozsah -128 až 127). Další, nikoliv však tolik zásadní problém, byl špatně spočtený kontrolní součet ICMP paketu. Původní a opravený kód týkající se sekvenčního čísla a identifikátoru je pro představu uveden v následujícím bloku.

```
if(icmp_pkt->icmp_type==ICMP_ECHOREPLY ||
   icmp_pkt->icmp_type==ICMP_ECHO ||
   icmp_pkt->icmp_type>ICMP_PARAMPROB){
    (*env)->CallVoidMethod(env,packet,setICMPIDMID,
                                (jshort)icmp_pkt->icmp_id >> 8,
                                (jshort)icmp_pkt->icmp_seq >> 8 );
}
```

Zdrojový kód 1 – původní kód knihovny

```
if(icmp_pkt->icmp_type==ICMP_ECHOREPLY ||
   icmp_pkt->icmp_type==ICMP_ECHO ||
   icmp_pkt->icmp_type>ICMP_PARAMPROB){
        (*env)->CallVoidMethod(env,packet,setICMPIDMID,
        (jshort) ((icmp_pkt->icmp_id >> 8) | (icmp_pkt->icmp_id << 8)),
        (jshort) ((icmp_pkt->icmp_seq>> 8) | (icmp_pkt->icmp_seq << 8)));
}
```

Zdrojový kód 2 – opravený kód knihovny [43], [44]

[22] Java Native interface

Na přiloženém CD nosiči je k dispozici zdrojový kód knihovny s již aplikovanými záplatami včetně opravy kontrolního součtu. K dispozici je i původní autorův web v offline podobě.

5.2.3 JNetPcap

Nový a stále se vyvíjející projekt, který se snaží odstranit nedostatky předchozích dvou. Knihovna je dostupná ve dvou verzích:

- open source – která je zdarma ke stažení a existuje podpora z řad členů komunity kolem tohoto projektu,
- profesionální – která je dostupná za poplatek včetně podpory přímo od vývojářů.

Oproti verzi Jpcap Keita Fuji nabízí více podporovaných protokolů. Namátkou podpora VoIP, RIP, ATM, … Ovšem její použití není tak intuitivní a je vhodné ji nasadit spíše na větších projektech. Rovněž je postavena nad libpcap/winpcap a autoři JNetPcap mají snahu v nových verzích svého produktu aktivně odrážet přidanou funkcionalitu do těchto knihoven.

5.2.4 Apache commons Net

Je balík nástrojů, jež usnadňuje práci se sítí pomocí dodávaných utilit a implementací různých protokolů, jako je např. FTP, TFTP, Telnet, Finger a další. V projektu jsou nástroje použity zejména pro práci s IP adresami jako je výpočet všech adres ze zadaného rozsahu nebo ověření, zda jde o platný vstupní formát adresy.

5.3 Knihovny pro vizualizaci

Knihoven pro práci s grafy existuje celá řada, ale liší se svým zaměřením a případně licenční politikou. Přesto, že se jejich orientace mohou překrývat, lze je rozdělit do několika kategorií.

Orientace na **datové modelování** – jde převážně o grafické frameworky poskytující funkcionalitu pro tvorbu aplikací, které vytváří různé druhy schémat nebo modelů jako jsou UML, BPM, ER diagramy případně další. Mezi zástupce této kategorie se řadí:

- YFiles,
- Picolo2D,
- JGraph.

Orientace na **datové struktury** – zaměření na optimální návrh datových struktur a algoritmů nad nimi. Algoritmy řeší různé problémy týkající se teorie grafů (problém obchodního cestujícího, hledání cest v grafu, …). Vizualizaci neumožňují vůbec nebo jen částečně:

- GMGraphLib,
- JGraphT,
- Annanas.

Orientace na **vizualizaci** – knihovny implementují některé z algoritmů řešící rozložení vrcholů ve 2D prostoru. Zejména se jedná o algoritmy řízené silou, které jsou doplněny o uživatelskou interakci:

- JUNG2,
- Prefuse.

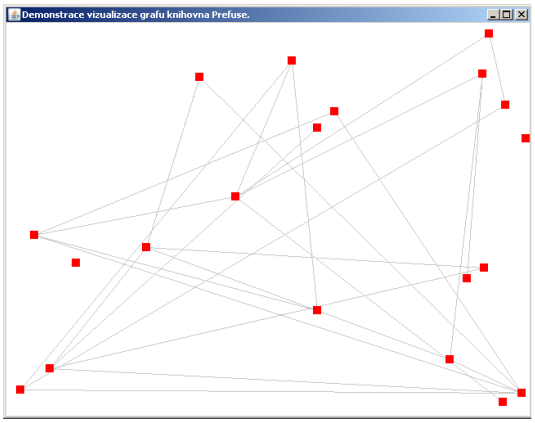
Jako nejvhodnější se k užití jevila poslední skupina orientovaná na vizualizaci a uživatelskou interakci. Oba projekty jsou distribuovány pod licencí BSD a jsou k dispozici:

- zdrojové kódy,
- dokumentace k API,
- tutoriály provázející začátečníky tvorbou,
- okomentovaná dema.

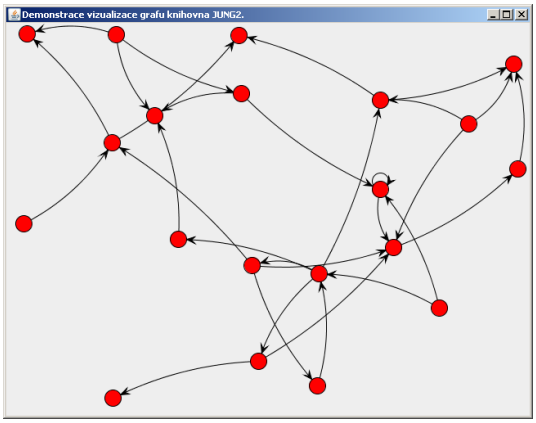
Jediný rozdíl, který byl patrný ihned při zjišťování informací o knihovnách, byl ten, že komunita kolem Prefuse je o něco aktivnější. Obě knihovny umožňují podobnou funkcionalitu a ani autoři[23] se přímo nepouštějí do srovnání, protože si váží a respektují práci toho druhého.

Rozhodujícím faktorem pro výběr byl tedy subjektivní pocit autora práce na základě složitosti (počet řádků kódu, API) vytvoření pomocné aplikace zobrazující jednoduchý graf.

Testovací aplikace vytvořila 20 vrcholů a vygenerovala 30 hran náhodně propojujících vrcholy. Kód pro plnění grafu ani nutné importy tříd nejsou do celkového počtu řádků potřebných k vytvoření a zobrazení grafu zahrnuty. Kód byl psán co nejvíce úsporným způsobem a to i na úkor čitelnosti (zejména u Prefuse).



Obrázek 37 – graf Prefuse



Obrázek 38 – graf JUNG2

[23] Prvotní autor Prefuse, Jeffrey Heer asistent profesora na univerzitě ve Standfordu a autor JUNG2 Joshua O¨Madadhian softwarový inženýr ve společnosti Google, jsou přátelé [49].

Tabulka 1 – porovnání Prefuse a JUNG2 co do počtu řádků kódu

Knihovna	Počet řádků
Prefuse	15
JUNG2	9

Při tvorbě vzorové aplikace se knihovna JUNG2 jevila jako mnohem vhodnější kandidát vzhledem k použitým programovým konstrukcím. Prakticky nemuselo docházet k jednořádkovým zápisům, které ztěžují čitelnost kódu. To by u Prefuse více než zdvojnásobilo nutný počet řádků, viz přílohy na straně 77.

V další kapitole bude podrobněji představena knihovna JUNG2, která poskytovala intuitivnější API a jevila se jako dobrý základ k vizualizaci naměřených dat a k tvorbě topologické mapy.

5.3.1 Java Universal Network/Graph 2

Jak bylo poznamenáno v odstavci výše, jde o framework pro práci s grafovou strukturou a její vizualizaci. Nabízí navíc jistou omezenou podporu i pro stromové struktury, ovšem jejich popisem se tento text nebude zabývat. Ke své činnosti používá ještě navíc podpůrné knihovny:

- Apache Commons Collections – utiliy, rozhraní pro práci s kolekcemi,
- JUnit – podpora pro pokrytí kódu testy,
- Colt – knihovna nabízející efektivní práci s výpočty.

Je postavena na čistě objektově orientovaném přístupu. Staví na principech dědičnosti a rozhraní. Což umožňuje snadné rozšíření stávajících struktur a vývoj nových, použitelných v již hotových algoritmech.

Datové struktury

HyperGraph<V,E> – bázové generické rozhraní definující funkcionalitu pro datový typ hyper graf, což je zobecněná podoba klasického grafu s tím rozdílem, že umožňuje, aby jedna hrana propojovala více vrcholů než dva (hrana není přímka, ale jakási forma množiny). Typové parametry *V* a *E* reprezentují konečný typ vrcholů a hran.

Graph<V,E> – generické rozhraní definující funkcionalitu pro základní operace s datovým typem graf jako je vkládání, odebírání, zpřístupnění kolekce vrcholů, hran, předchůdců nebo následníků, získání informací o koncových bodech patřičné hrany a další. Toto rozhraní dědí z bázového *HyperGraph* a tvoří tak základ pro práci s grafy.

DirectGraph<V,E> – rozhraní, které dědí z rozhraní *Graph* a slouží jako signalizace pro implementující třídy, že se jedná o orientovaný graf, nikterak nedoplňuje funkcionalitu.

UndirectedGraph<V,E> – totéž co předešlé rozhraní s tím rozdílem, že značí neorientovaný graf.

K dispozici je nadále abstraktní datový typ, který implementuje rozhraní *Graph*:

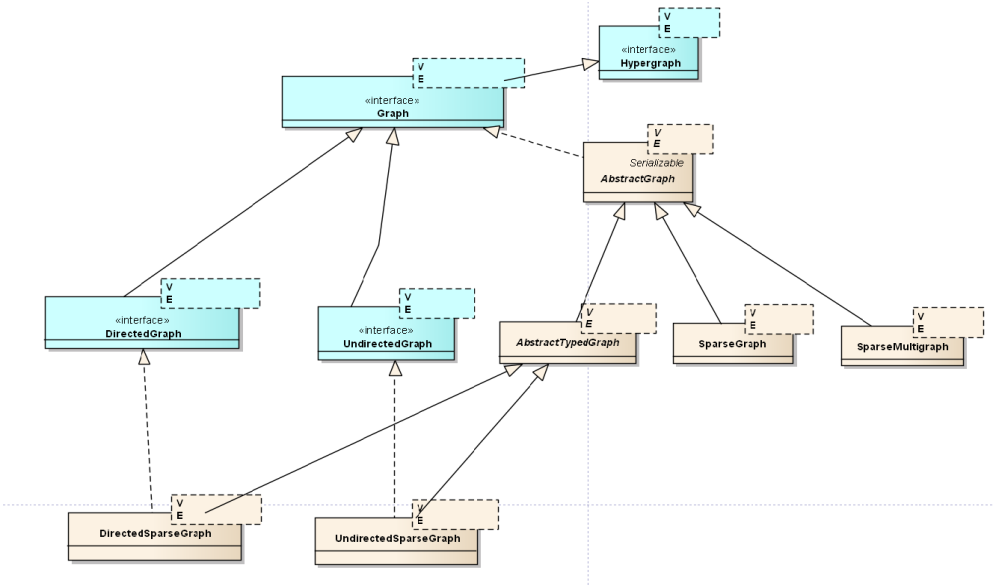
AbstractGraph<V,E> – sloužící jako základní bázový typ pro třídy:

- AbstractTypeGraph – dovoluje pouze hrany stejného typu,
- SparseGraph – dovoluje oba typy hran jak orientované, tak neorientované,
- SparseMultiGraph – stejné jako *SparseGraph* a navíc umožňuje paralelní hrany.

Mezi použitelné implementace tříd pro práci s grafy patří:

- DirectedSparseGraph – orientovaný graf nedovolující paralelní hrany,
- DirectedSparseMultiGraph – orientovaný graf dovolující paralelní hrany,
- UndirectedSparseGraph – neorientovaný graf nedovolující paralelní hrany,
- UndirectedSparseMultiGraph – neorientovaná graf dovolující paralelní hrany.

Pro lepší představu hierarchických vazeb bude následovat class diagram, ve kterém jsou záměrně vynechány metody, atributy a některé třídy/rozhraní. Každý graf totiž nabízí verze pro orientované a neorientované hrany, přičemž tyto lze dále rozlišit na multigraf, který obsahuje možnost paralelních hran.



Obrázek 39 – diagram tříd pro práci s grafy

Z konceptu abstraktní třídy, která implementuje rozhraní, pramení jedna nesporná výhoda a tou je možnost přidat funkcionalitu do již implementovaných tříd bez nutnosti je všechny modifikovat Pokud by třídy implementovaly pouze rozhraní, vznikl by zde problém s přidáním nové funkcionality, v takovém případě by bylo nutné doplnit požadovanou funkci jednak do rozhraní, ale i do všech tříd, které toto rozhraní implementují. Nabízí se otázka proč nepoužít pouze abstraktní třídu, důvod je ten, že v kombinaci s rozhraním není

programátor v případě rozšíření tolik svázán původním návrhem jako s čistě abstraktní třídou.

Obdobného konceptu využívají tvůrci jazyka Java 8. Při doplnění podpory pro lambda výrazy bylo nutné zasáhnout i do stávajících kódů rozhraní, která implementovala např. kolekce a v případě takového zásahu by to znamenalo upravit všechny třídy, které dané rozhraní implementují, což je nepřípustné. Autoři to ovšem vyřešili tak, že dovoluji, aby funkce v rozhraní mohla definovat i tělo, čímž se mimo jiné do Javy dostane i vícenásobná dědičnost [45].

Algoritmy

JUNG2 nabízí k dispozici řadu připravených algoritmů řešících grafové úlohy jako např.

- výpočet toků v grafu,
- nalezení nejkratší cesty,
- algoritmy pro sociální sítě,
- hodnotící algoritmy,
- grafové generátory, filtry a další.

Knihovna hojně využívá ve svých algoritmech a datových strukturách myšlenku návrhových vzoru Factory a Transformer. Generickou podobu rozhraní pro oba vzory nabízí dříve představená knihovna Collections z projektu Apache Commons.

Transformer<I,O> – generické rozhraní s jednou metodou *O transform(I),* která na svém vstupu přijímá objekt typu *I*, který transformuje na výstupní objekt typu *O*. Užitečné při návrhu obecných algoritmů a struktur.

```
public static class VertexLabellerTransformer implements Transfor-
mer<Host, String> {
        @Override
        public String transform(Host input) {
            return input.getInetAddress();
        }
}
```

Zdrojový kód 3 – návrhový vzor Transformer

Ukázka jednoduchého transformeru *VertexLabellerTransformer*, který jako svůj vstupní parametr přijímá třídu *Host* a vrací jednoznačný identifikátor IP adresu typu *String*.

Factory<T> – rozhraní definující jednu metodu *T create()* bez parametru, která vrací objekt typu *T*. Vhodné pro zautomatizované vytváření objektů.

```
public class LinkFactory implements Factory<Link>{
        private static int id = 0, cost =0, rtt  = 0;
        private static String name = "";
        private static final LinkFactory instance = new LinkFactory();
        private LinkFactory() {}
        public static LinkFactory getInstance() {
            return instance;
        }
        @Override
        public Link create() {
            return new Link(name,id){{setCost(cost);setRtt(rtt);}};
        }
}
```

Zdrojový kód 4 – návrhový vzor Factory

Jednoduchá třída *LinkFactory* demonstrující ukázku návrhového vzoru Factory a Singleton. Jedná se o statickou tovární třídu pro instance typu *Link*.

Vizualizace

Autoři knihovny vsadili na koncept jednoduchosti a pro vykreslení základního grafu stačí minimum kódu. Vše, co je potřeba k zobrazení, je:

1. graf samotný, naplněný daty,
2. vhodný výběr implementace algoritmu rozložení vrcholů v ploše,
3. vizualizační komponenta (nějaká podoba třídy JPanel),
4. bázová GUI komponenta, nejčastěji Swing JFrame.

Algoritmy rozložení

K dispozici je následující aparát pro práci s rozložením. Většina algoritmů byla představena v teoretické části, bude tedy následovat pouze stručný přehled.

- BaloonLayout – stromově orientovaný algoritmus.
- CircleLayout – rozložení vrcholů grafu do kruhu.
- DAGLayout – stromově orientovaný algoritmus vhodný pro acyklické orientované grafy.
- FRLayout – implementace algoritmu řízeného silou Fruchterman-Reinghold.
- ISOMLayout – implementace Meyersova samo-organizujícího algoritmu, vycházejícího z oblasti neuronových sítí.
- KKLayout – implementace algoritmu řízeného silou Kamada-Kawai.
- RadialTreeLayout – orientace na stromy.
- SpringLayout – implementace algoritmu řízeného silou, vycházející z principu pružin.
- StaticLayout – uživatelem definované rozložení.

Vizualizační komponenty

Jedná se o komponenty použitelné při tvorbě GUI, které umožňují zobrazit graf. Typicky se jedná o potomky dědící ze třídy JPanel.

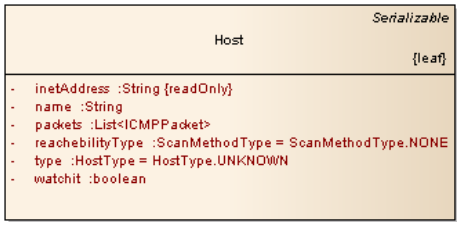
- BasicVisualizationServer – jednoduchá komponenta absentující uživatelskou interakci.
- VisualizationViewer – třída doplňující *BasicVisualizationServer* o uživatelskou interakci pomocí myši.

5.4 Třídy definující síťovou topologii

Tato kapitola řeší nalezení vhodné vazby mezi třídou a objektem reálného světa (oblast počítačových sítí). Třídy pak dané objekty v modelu zastupují. Návrh se snaží o jednoduchost a nezanedbání potřebných detailů. U většiny class digramů byly vypuštěny metody, protože se jedná o datové třídy neřešící žádnou důležitou logiku. Jejich význam je čistě zástupného charakteru.

5.4.1 Uzel

Uzlem je v modelu chápán aktivní síťový prvek z počítačové sítě, který je schopen komunikovat prostřednictvím TCP/IP protokolů. Z pohledu teorie grafů se jedná o vrchol. Uzel je v kódu reprezentován třídou **Host** v baličku *net.data* a každý uzel může nabývat určitého typu. Jednotlivé typy jsou definovány jako interní veřejný statický výčtový typ **HostType** uvnitř třídy *Host.*



Obrázek 40 – důležité atributy třídy Host

InetAddress – jednoznačný identifikátor typu *String*, sloužící jednak jako popisný atribut (internetová adresa) a rovněž jako klíč ve struktuře *HashMap*, ve které se uchovávají všechny uzly (vrcholy grafu). Obecně není sice doporučeno používat typ *String* jako klíč ve struktuře *HashMap*, protože jeho rozsah hodnot je mnohem větší než např. rozsah celých čísel a tím se zvyšuje i riziko kolize, nicméně v tomto případě je platnost omezena na 15 znaků, čímž se eliminuje vznik kolizí.

Name – doménové jméno získané reverzním překladem z IP adresy. Tato položka je uchovávána z důvodu šetření výpočetní kapacity na neustálé dotazování DNS serveru. Dotaz je proveden pouze jednou. Nadále je hodnota čtena z paměti.

Packets – seznam paketů, který se naplní ve fázi detekce dostupných uzlů. Při detekci dochází k zaslání uživatelem zvoleného počtu dotazů na definovaný cíl a pro dotazy, ke kterým přijde odpověď, vznikne příslušný záznam v této struktuře (seznam). Na základě čehož lze vyvodit statistické závěry, spočítat ztrátovost, případně čas odezvy.

ReachabilityType – představuje typ dostupnosti. Proměnná může nabývat hodnot:

- ICMP – uzel reaguje na dotazy skrze protokol ICMP,
- UDP – uzel reaguje na dotazy skrze protokol UDP,
- BOTH – kombinace obou výše zmíněných typů.

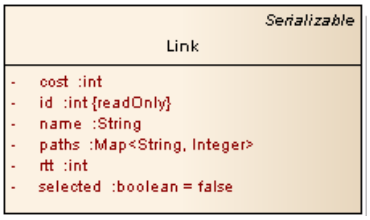
Type – rozlišují se zde následující typy uzlů.

- Router – uzel, přes který vede cesta, jinak řečeno má takový vrchol v grafu svého předchůdce a následníka.
- Unknown – neznámé zařízení, jehož identita se nepodařila prokázat.
- First in trace – uzel, ze kterého probíhá detekce, obvykle koncová uživatelská stanice. Tento typ by se měl v modelu nacházet právě jednou.
- Last in trace – uzel, který představuje koncové zařízení, jež se při detekci jevilo jako aktivní.
- Broken – uzel dočasně vyřazen z provozu.

Watchit – proměnná indikující stav, zda má být uzel zařazen do sledování, či nikoliv.
V případě nastavení na *false* nebude prvek zařazen do procesu sledování.

5.4.2 Linka

Jde o spojnici mezi dvěma uzly v modelu. V reálné síti může reprezentovat propojovací médium, jehož podobu ovšem nelze při detekci dostatečně ověřit. Z pohledu teorie grafů se jedná o hranu. Ve zdrojovém kódu je tento objekt zastupován třídou *Link*.



Obrázek 41 – důležité atributy třídy Link

Cost, rtt – představují cenu/váhu a čas zpoždění na lince. Tyto atributy nejsou v práci využity a slouží jako možnost rozšíření.

Id – jednoznačný číselný identifikátor linky sloužící jako klíč v datové struktuře *HashMap*.

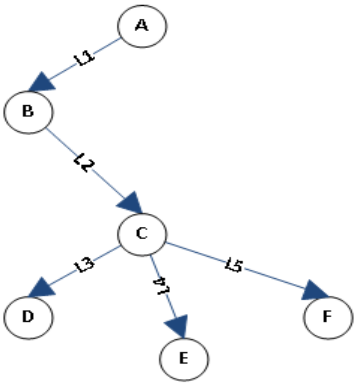
Name – popisný atribut představující uživatelem definované jméno linky.

Selected – značí, zda je daná linka součástí trasy, která se má uživateli zobrazit.

Paths – struktura uchovávající trasy, které vedou přes zadanou linku. Zároveň je udržován i počet chybějících uzlů pro každou cestu a linku.

Může, nastat situace, že některý z uzlů při detekci trasy neodpoví v řádném čase nebo vůbec. V takovém případě se jejich počet uloží do této struktury spolu s trasou, pro kterou

toto měření odhalilo nedostatky. Pokud je tento údaj nenulový, je při vizualizaci trasy zobrazen na dané spojnici a reprezentuje počet chybějících uzlů. Každá trasa je identifikována vždy svým cílovým uzlem.



Obrázek 42 – fiktivní topologie

Tabulka 2 – trasy vedoucí přes linky

Linka	Trasa
L1	D, E, F
L2	D, E, F
L3	D
L4	E
L5	F

5.4.3 Cesta

Cestu lze vyjádřit seznamem po sobě jdoucích uzlů nebo linek vedoucích z místa **Ah** do místa **Bh**. Ah reprezentuje uživatelskou stanici, ze které probíhá detekce a Bh pak uživatelem zvolený cílový uzel. **Model dovoluje, aby do koncového uzlu vedla pouze jediná cesta.**

Cesta je zastoupena třídou **Path** a pracuje s vrcholy, pro které využívá pomocnou obalovou třídu **PathElement** uchovávající dva vrcholy a počet nenalezených uzlů mezi nimi.

Obrázek 43 – důležité atributy třídy Path

KeyPath – jednoznačný identifikátor trasy v podobě IP adresy cílového uzlu.

Elements – pomocné elementy odrážející chybějící uzly mezi dvěma vrcholy.

Obrázek 44 – obalová třída PathElement

Diff – počet chybějících uzlů mezi dvěma vrcholy.

Pair – struktura uchovávající dva vrcholy.

Koncept pomocných tříd jako je např. *PathElement* byl pojat jako návrhový vzor *Messanger*, který je vhodný pro přenos více atributů v jedné obalové třídě. Takováto třída vystavuje své atributy jako veřejné a jedinou inicializaci těchto hodnot je možné provést pouze prostřednictvím konstruktoru. Po inicializaci je možné hodnoty pouze číst. Návrhový vzor nedefinuje žádnou dodatečnou logiku nad svými daty.

Mechanismus detekce chybějících uzlů funguje na principu odhalení chybějících paketů s danou hodnou TTL. Po detekci trasy se provede její normalizace, při které se odhalené díry zaznamenají do pomocných struktur, s kterými se pak nadále pracuje.

5.4.4 Topologie

Topologií je zde míněn model počítačové sítě, jež obsahuje množinu uzlů propojených pomocí linek. Z pohledu teorie grafů se jedná o graf.

Jako vhodná datová struktura odrážející graf byla zvolena třída **DirectedSparseGraph**, která reprezentuje orientovaný graf nedovolující více násobné hrany. Orientovaný graf byl zvolen z důvodu, že orientace hrany umožňuje vyznačit tok paketů od zdroje k cíli. Vzhledem k dostupným zdrojovým kódům bylo možné nahlédnout na vnitřní implementaci dané třídy, která je následující.

```
Map<V, Pair<Map<V,E>>> vertices = new HashMap<V, Pair<Map<V,E>>>();
Map<E, Pair<V>> edges = new HashMap<E, Pair<V>>();
```

Zdrojový kód 5 – struktury uchovávající data pro třídu DirectedSparseGraph

Vrcholy a hrany jsou uloženy ve struktuře *HashMap* a organizovány na základě předefinované funkce *hashCode()* objektu. Každý z vrcholů si nadále udržuje seznamy (ve skutečnosti opět struktury *HashMap*) svých předchůdců a následníků kvůli rychlejšímu přístupu.

Hrany jsou rovněž udržovány ve struktuře *HashMap* a identifikovány stejně jako vrcholy na základě předefinované funkce *hashCode()* a navíc si udržují odkazy na vrcholy, kterými jsou určeny.

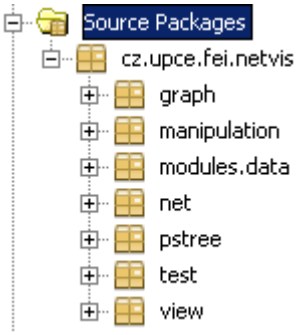
Pair<T> je jednoduchá třída uchovávající dvě hodnoty téhož typu, u nichž záleží na pořadí. Obsahuje dva privátní atributy *first* a *second*. Jedná se o knižní implementaci z knihovny JUNG2.

5.5 Architektura

Architektura aplikace je rozdělena na tři hlavní vrstvy.

- Síťová – obslužné třídy jsou dostupné v baličku **cz.upce.fei.netvis.net** a starají se především o:
    - detekci uzlů,
    - detekci tras,
    - ověření dostupnosti vybraných uzlů.

- Vizualizační – obslužné třídy jsou dostupné v baličku **cz.upce.fei.netvis.view** a jejich hlavní náplní je:
    - grafická reprezentace naměřených dat,
    - uživatelská interakce.
- Datové struktury – tvoří podklad pro vizualizaci a jsou obsaženy v balíčku **cz.upce.fei.netvis.graph**. Balíček dále obsahuje implementace pomocných tříd dle návrhových vzorů Facotry a Transformer.



Obrázek 45 – bázové baličky projektu

Dále balíček **cz.upce.fei.netvis.manipulation** obsahuje kódy potřebné pro vstupně výstupní operace, které jsou v aplikaci použity.

Baliček **cz.upce.fei.netvis.modules** zaštiťuje logiku práce s moduly/příkazy které může uživatel zadávat a vyvolávat nad vybranými uzly.

Mezi méně důležité balíčky patří **cz.upce.fei.netvis.test** obsahující testovací kódy a **cz.upce.fei.netvis.pstree**, který obsahuje částečnou implementaci prioritního vyhledávacího stromu.

Při studiu zdrojových kódů bylo odhaleno, že implementace knihovních funkcí pro výběr objektů v grafickém panelu (na základě uživatelského výběru) používají na zjišťování nejbližšího objektu ke kurzoru myši **algoritmus se složitostí O(n)**. Kde n je počet vrcholů. Jinými slovy musí pokaždé projít všechny vrcholy, aby zjistil, který je nejblíže, což u menšího počtu objektů nemusí činit potíže, nicméně v případě většího počtu zajisté bude.

```
public V getVertex(Layout<V,E> layout, double x, double y, double max-
Distance) {
  double minDistance = maxDistance * maxDistance;
  V closest = null;
      while(true) {
          try {
           for(V v : layout.getGraph().getVertices()) {
            Point2D p = layout.transform(v);
            double dx = p.getX() - x;
            double dy = p.getY() - y;
            double dist = dx * dx + dy * dy;
            if (dist < minDistance) {
                minDistance = dist;
                closest = v;
            }
         }
         break;
     } catch(ConcurrentModificationException cme) {}
}
return closest;
}
```

Zdrojový kód 6 – algoritmus se složitostí O(n)

Vhodnější je využití struktury, která nějakým způsobem omezí potřebnou množinu dat pro průchod, čímž PST[24] rozhodně je. Ovšem úplná implementace stromu nebyla dokončena ani začleněna do finální verze, protože se během testování nepodařil prokázat dostatečný vliv na výkon (testováno bylo se stovkami uzlů).

5.5.1 Koncept tříd řešících detekci

Návrh detekčních nástrojů byl pojat modulárně, to znamená, že konečný nastroj je spojen z několika dílčích objektů a dohromady tvoří celek. Základní stavební bloky pro všechny detekční nástroje tvoří třídy **Captor** a **Sender**, přičemž každá z nich se stará pouze o svou funkcionalitu.

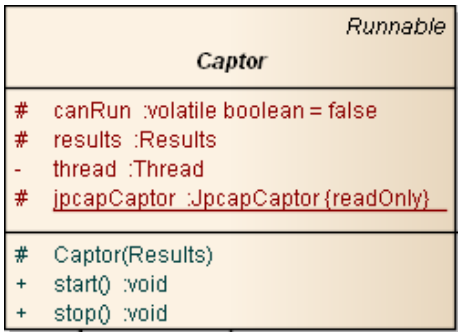
Jak již název napovídá třída *Captor* se stará pouze o zachytávání paketů zatímco *Sender* obstarává odesílání. Až ve finální fázi implementace nástroje jsou tyto třídy využity pro definici konkrétního chování. Při návrhu tříd byla snaha držet se minimalizačního pravidla, aby třída obsahovala jen ten nejnutnější kód a starala se pouze o to, co jí náleží.

Třída *Captor*

Abstraktní třída zapouzdřující základní funkcionalitu pro práci se zachytáváním paketů. Obaluje instanční proměnnou z knihovny JPcap třídy *JpcapCaptor*, která nastavuje obecný filtr na požadovaný formát zachytávaných paketů a otevírá buffer síťového adaptéru pro možnost čtení.

[24] Priority Search Tree – jde o strom, který má haldové uspořádání dle souřadnice **y** a dle **x** splňuje kritéria binárního vyhledávacího stromu. Umožňuje intervalové i bodové vyhledávání.

Instance třídy *JpcapCaptor* je instanciována ve statickém inicializátoru této třídy, což umožňuje v celém kódu používat pouze jedinou instanci tohoto druhu (tento princip nemá suplovat návrhový vzor Singleton pro třídu *Captor*). S otevřením a zavřením je spojena jistá režie, která je tímto eliminována na minimum a navíc je zaručeno, že po dobu běhu aplikace bude práce všech nástrojů vykonávána nad stejnou kopii bufferu.



Obrázek 46 – třída Captor

Idea při návrhu byla taková, že každý nástroj bude obsahovat jedno vlákno, které pakety zachytí a pak několik vláken, které budou pakety odesílat. Přičemž nástroje (vlákna samozřejmě ano) nebudou schopny souběžné činnosti, ale sériové, ale ani to jejich charakter nedovoluje, protože, jeden čeká na výsledky druhého. V opačném případě by bylo nutné zajistit korektní čtení z bufferu pro více vláken. K tomu by však v aplikaci mohlo dojít pouze v případě, že se uživatel pokusí vyvolat detekci nových uzlů v okamžiku, kdy běží subproces pro detekci dostupnosti uzlů (watchdog), tomu je ale v aplikaci programově zabráněno.

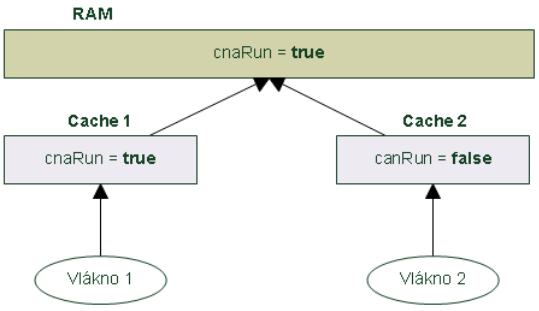
Při studiu zdrojových kódů knihovny autor sám, omezuje počet možných instancí na 255, viz konstanta *MAX_NUMBER_OF_INSTANCE* ve zdrojovém souboru[25] abstraktní třídy *JpcapInstanc.java.*

Kód statického inicializátoru je volán v době, kdy JVM class loader zavádí třídu do paměti, takže je zajištěna i více vláknová bezpečnost v případě, že by se více vláken pokoušelo vytvořit instanci (volání class loaderu je vláknově bezpečné). Každá třída je v paměti identifikována svým názvem a class leaderem, který ji do paměti natáhl. Toto řešení ovšem není odolné situaci, kdy se danou třídu pokusí načíst více class loaderu (častý jev při vývoji EE aplikací). Při návrhu se s více class loadery nepočítá, nicméně řešení lze nalézt pomocí java reflection API [46].

Jak již bylo naznačeno dříve, třída je schopna běžet jako samostatné vlákno, takže lze realizovat zachytávání paketů nezávisle na ostatních subprocesech. Obsahuje základní operace pro zahájení a ukončení chodu vlákna. Zároveň obsahuje atribut ***canRun*** signalizující běh vlákna, který je označen jako **volatile.** To z důvodu, aby si vlákno neudržovalo tuto proměnou ve své cache. Jinak by mohlo dojít k situací, ve které se pokus o zastavení vlákna

[25] Kompletní zdrojový kód je dostupný na přiloženém CD nosiči.

nikdy nezdaří. Důvodem je fakt, že jedno vlákno pracuje se svou kopii proměnné v cache, přičemž druhé se marně snaží pracovat se svou kopií. Takto obě pracují nad stejnými daty.



Obrázek 47 – využití paměti cache vlákny

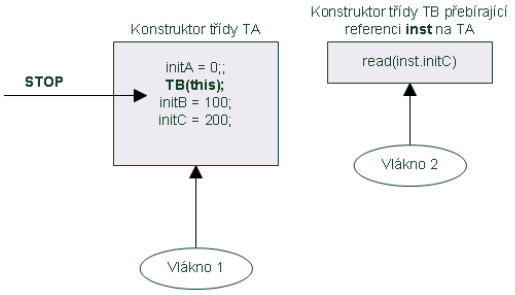
V tomto konkrétním případě se vlákno 1 nemusí nikdy ukončit na žádost vlákna 2, jak bylo poznamenáno v odstavci výše.

Prvotní myšlenka byla vytvořit instanci (nikoliv spuštění) vlákna přímo v konstruktoru třídy. Java nabízí pro práci s vlákny třídu ***Thread,*** která přebírá jako svůj parametr referenci na třídu implementující rozhraní ***Runnable*** , čímž třída *Captor* je viz Obrázek 46 – třída Captor.

```
protected Captor(Results results) {
        thread = new Thread(this, "Captor thread:");
        this.results = results;
    }
```

Zdrojový kód 7 – chybná inicializace v konstruktoru třídy

Ačkoliv se tato úvaha může jevit na první pohled jako korektní, není tomu tak. Není totiž obecně dobrou praktikou ve více vláknovém prostředí zveřejňovat instanční ukazatel **this** v průběhu konstrukce mimo objekt. Důvod je prostý, v průběhu konstrukce může instance z jiného vlákna přistupovat k ne úplně zkonstruovanému objektu.



Obrázek 48 – vliv zveřejnění reference this v průběhu konstrukce objektu

V případě, že vlákno 2 začne pracovat s objektem třídy TA dříve, než je zkonstruován, je takové chování nedefinované, což zajisté není žádoucí jev.

U instančních metod lze tento jev v případě potřeby ošetřit pomocí synchronizovaných bloků, ale u konstruktoru je nutné se tomu vyhnout tím, že nebezpečný kód odkloníme do příslušné metody. Kód byl na základě tohoto poznatku modifikován tak, aby vyhovoval správným principům pro více vláknové programování.

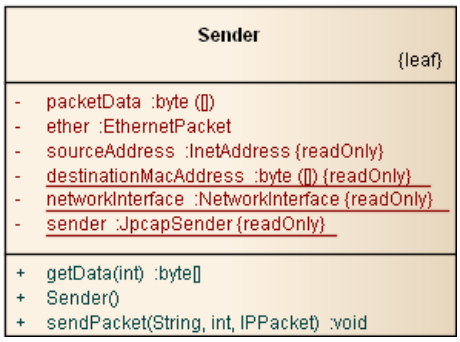
Každý potomek třídy *Captor* umožňuje nadále ukládání potřebných mezivýsledků do datové položky typu *Results*, což je rozhraní. V aplikaci existuji dvě implementace:

- ResultsSolverProbe,
- ResultsSolverPing.

Každá implementace je charakteristická mírně odlišným způsobem chování, které je ovlivněno nástrojem, jež objekt využívá.

Třída *Sender*

Jednoduchá třída, která zapouzdřuje instanci knihovní třídy *JpcapSender* zajišťující funkcionalitu odesílání paketů.



Obrázek 49 – třída Sender

Třída stojí na podobném principu jako předešlá *Captor*. Ve statickém inicializátoru se nejprve:

- vytvoří nová instance třídy *JpcapSender* na základě uživatelem definovaného síťového rozhraní reprezentovaného třídou *NetworkInterface*,
- zjistí se MAC adresy síťové karty a výchozí brány, které jsou použity dále při konstrukci ethernetových rámců.

Důvody pro statickou inicializaci těchto položek jsou shodné s těmi ve třídě *Captor*. Navíc získání MAC adresy výchozí brány (fyzická adresa) obnáší jistou dodatečnou časovou režii, protože je nutné nejprve vytvořit fiktivní spojení na zadaný server a odchytit příchozí paket, ze kterého je již možné adresu získat. Přímo knihovna (ani Java) nenabízí žádný jednodušší mechanismus, proto se o detekci MAC adresy brány stará pomocná statická metoda *getGatewayMAC()* obsažena ve třídě *NetworkUtils*. Ta zaštiťuje i další pomocné funkcionality využívané v aplikaci. Zdrojovou MAC adresu je možné získat z instance třídy *NetworkInterface*.

```
instance = JpcapCaptor.openDevice(networkInterface, 100, false, 50);
instance.setFilter("tcp and host " +
Inet4Address.getByName("server.cz").getHostAddress(), true);
new URL("http://server.cz").openStream().close();
Packet packet = instance.getPacket();
  if (packet != null) {
    return ((EthernetPacket) packet.datalink).dst_mac;
  }
```

Zdrojový kód 8 – získání MAC adresy

Pokud se detekce nepodaří, je vyhozena výjimka. Kód je zde uveden z důvodu, že může za jistých okolností působit jako zdroj problémů. V případě nedostupnosti dotazovaného serveru nebude možné MAC adresu získat a aplikace selže. Stojí za zvážení, zda v budoucnu nenavrhnout sofistikovanější způsob řešení. Nicméně, pokud je server dostupný, nehrozí žádné riziko.

O samotné odesílání se stará synchronizovaná instanční metoda *sendPacket()* přebírající jako své parametry:

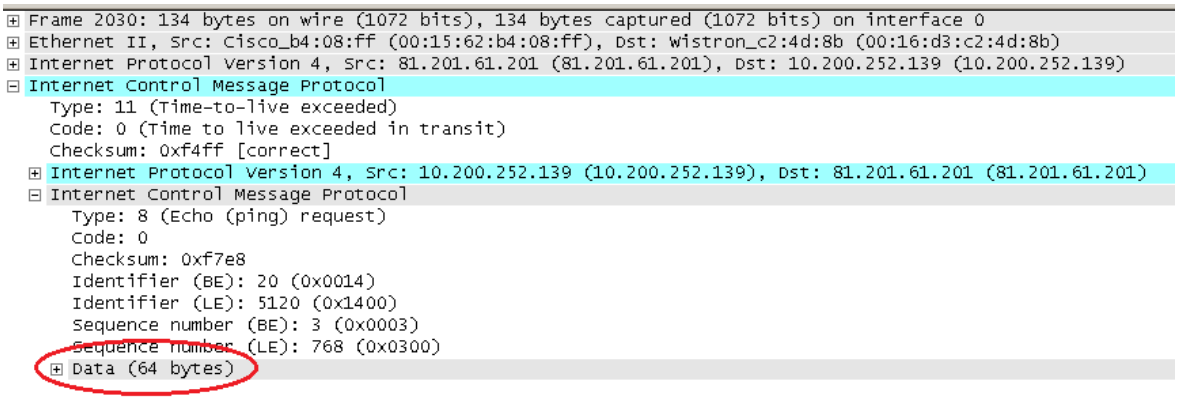
- IP adresu uzlu, na kterou se paket odešle,
- hodnotu TTL určující životnost paketu,
- samotný paket, který může být typu ICMP nebo UDP.

Uvnitř metody se zkonstruuje ethernetový rámec a nastaví se potřebné pole v IP záhlaví. Metoda je synchronizována z důvodu ochrany při použití ve více vláknech.

Během vývoje a testování implementovaných nástrojů bylo zjištěno, že některé implementace protokolu IPv4 nedodržují specifikaci RFC 792[26] (ani v úpravách 4884, 6633, 6981, 950 nebylo uvedeno jinak), která definuje, že ICMP paket typ 11 obsahuje ve své datové části původní IP záhlaví a **8 bytů** neseného paketu (ten, co způsobil potíže) kvůli identifikaci v aplikaci [47].

V rámci testu bylo však odhaleno něco jiného. Byl zaslán určitý počet ICMP paketů o velikosti IP záhlaví + ICMP záhlaví + 64 B dat na vybrané uzly s cílem podnítit je k vygenerování zprávy ICMP typu 11. Některé odpovědi však vykazovaly zdání, že se vrací nikoliv 8 bytů původního paketu, jak lze očekávat, nýbrž celý paket viz Obrázek 50 – porušení specifikace RFC 792.

[26] Definice konceptu ICMP protokolu z roku 1981.



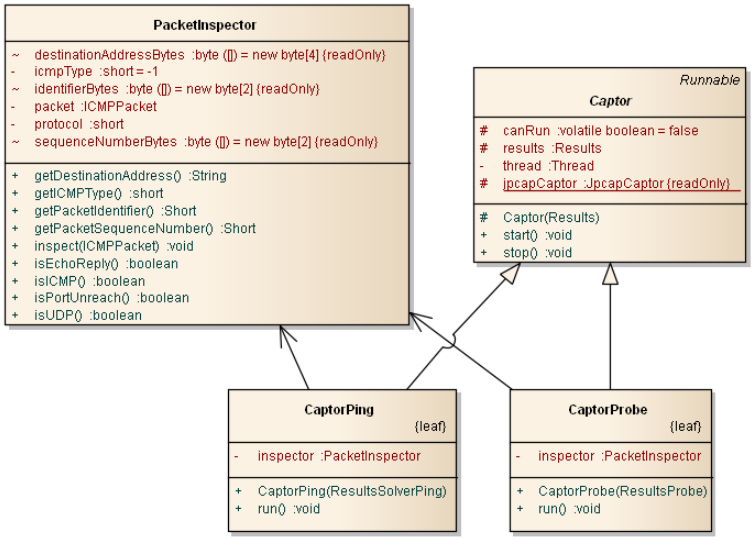
Obrázek 50 – porušení specifikace RFC 792

Toto chování bylo zpozorováno u implementace na operačním systému Linux. Například směrovače od společnosti Cisco zasílaly korektní zprávy (IP záhlaví + 8 bytů). Toto chování bude nejspíš typické i pro ostatní typy zpráv, jež v sobě nesou původní paket, nicméně ty nebyly dále testovány. Prvotní záměr totiž počítal s využitím této datové části k přenosu jistých druhů informací, ale ukázalo se, že toto chování nelze očekávat u všech zařízení, proto bylo od návrhu upuštěno.

Specializace třídy *Captor*

Třída v návrhu obsahuje dvě své specializace, jedná se o:

- *CaptorPing* – zachytávaní dat pro nástroj použitý při detekci uzlů,
- *CaptorProbe* – zachytávaní dat pro nástroj použitý při odhalování trasy.



Obrázek 51 – specializace třídy Captor

Třída **PacketInsector** se používá pro vyšetření různých polí přijatého paketu, především však pro zisk **sekvenčního čísla** a **identifikátoru** pro možnost sdružování souvisejících paketů.

V případě vyšetřování navrátivších se paketů (ICMP typ 11) nesoucích jako zdroj chyby ICMP typ 8 byla situace jednoduchá, stačilo přečíst z:

- ICMP záhlaví dvě dvou bytové pole *sequence number* a *identifier*,
- IP záhlaví 4 bytové pole *destination IP address*.

A takto získat patřičné hodnoty.

Ovšem při vyšetřování paketů nesoucích zprávu indikující problém s protokolem UDP, byla situace složitější. V takovém případě se totiž vracela zpráva ICMP typ 3 s kódem 3 nesoucí jako problémový paket UDP (IP záhlaví a 8 bytů z UDP). Žádný z těchto příchozích paketů však neobsahoval informace o sekvenčním čísle nebo identifikátoru a nebylo možné takový paket nikam přiřadit. Jediné, co bylo možné přečíst, byla cílová destinace, na kterou paket mířil. Bylo tedy nutné nějakým způsobem přepravit potřebné informace, které se cestou neztratí. V úvahu připadalo několik možností:

UDP záhlaví – uložit sekvenční číslo a identifikátor jako kombinaci do pole pro zdrojový nebo cílový port, což by ovšem znemožnilo uživateli volit port ručně. Od této verze bylo upuštěno.

UDP data – jak bylo popsáno výše, na tuto položku nelze spoléhat, protože ne vždy se datová část vracela celá, případně vůbec. I od této verze bylo upuštěno kvůli nespolehlivosti.

Standardní IP záhlaví – vzhledem k tomu, že se vracelo i celé IP záhlaví, nabízela se možnost využití některého z polí přímo v něm. Jako vhodné, co do velikosti, se zdálo 4 bytové pole **identification**, mající primární účel jako unikátní identifikátor fragmentů, které vznikly při fragmentaci, a bude z nich v cíli sestaven původní paket. Dle RFC 4413 není přiřazení hodnot tomuto poli přesně určeno, pouze uvádí, že musí být unikátní pro zdroj, cíl a protokol, což indikuje, že lze pro přiřazení použít více než jeden způsob.

Při návrhu detekčních nástrojů se uvažuje, že při každém odeslaném paketu (pokus o navázání spojení na příslušný port) se provede inkrementace hodnoty sekvenčního čísla reflektující pořadí odesílaného paketu. Čímž by byla zajištěna unikátnost pro danou trojici zdroj, cíl a protokol v případě odeslání jednoho paketu. Aplikace ale umožňuje odeslat na tentýž zdroj několik stejných paketů, čímž by v případě fragmentace mohlo dojít k promíchání fragmentů z různých paketů a k narušení integrity nesených dat. Nicméně nesená data nemají žádný užitný charakter. Slouží pouze jako výplň.

Navíc zasílané pakety mají obvykle délku 32 B[27], je tedy předpoklad, že k fragmentaci docházet nebude. Tohoto nedostatku si je autor práce vědom a možné řešení může poskytovat volitelné pole IP záhlaví. [48]

[27] Standardní IP záhlaví (20 B) + ICMP záhlaví (8 B) + datová výplň (4 B).

Volitelné položky IP záhlaví – jistou možnost k zamyšlení mohou nabízet i volitelné položky IP záhlaví. Tato možnost je zde uvedena jako možný směr budoucího vylepšení aplikace a nebyla důkladněji zkoumána.

Na základě předešlých úvah bylo pro přenos sekvenčního čísla a identifikátoru zvoleno pole **identification** ve standardním IP záhlaví.

Celý koncept tvorby jednotlivých nástrojů se kvůli efektivitě opírá o funkcionalitu vláken a stojí na principech návrhového vzoru využívající **fond vláken**[28], který obsahuje části jako:

- frontu (**task queue**), kam jsou řazeny jednotlivé úlohy,
- fond znovupoužitelných vláken (**worker threads**), které řeší úlohy z fronty,
- dozorčí vlákno (**solver thread**), které vše řídí a na konci zpracuje výsledky.

5.5.2 Detekce uzlů

O samotné objevování uzlů ze zadaného rozsahu se stará nástroj reprezentovaný třídou **SolverPing**, která je designovaná jako jednoúčelová, to znamená, že instance není znovupoužitelná a v případě zopakování akce je nutné vytvořit instanci novou. To pramení z rozšíření třídy **SwingWorker**, která je součástí JDK a rovněž použitelná jednoúčelově. Použití této kombinace tříd bylo voleno ze dvou důvodů:

- bezpečný přístup z jiného vlákna ke komponentám Swingu za účelem aktualizace GUI komponent v průběhu výpočtu,
- odklonění výpočetně náročného úkolu mimo EDT[29], který by jej mohl brzdit a vznikl by dojem, že aplikace dočasně zamrzla (než skončí úloha, která blokuje zpracování událostí v GUI).

Uživatel je tak díky tomuto řešení informován o průběhu detekce pomocí progres barů a aplikace tak budí „živější dojem“.

SolverPing přebírá v konstruktoru **pole IP adres**, jejichž dostupnost má ověřit a **parametry** detekce:

- velikost datové části odesílaného paketu,
- hodnotu TTL určující dobu životnosti paketu v síti,
- počet odeslaných paketů na daný cíl,
- zpoždění mezi odesíláním jednotlivých paketů na konkrétní uzel,
- typ použité metody UDP, ICMP nebo obojí,
- počet vláken, jež budou řešit úlohy.

[28] Thread pool pattern.

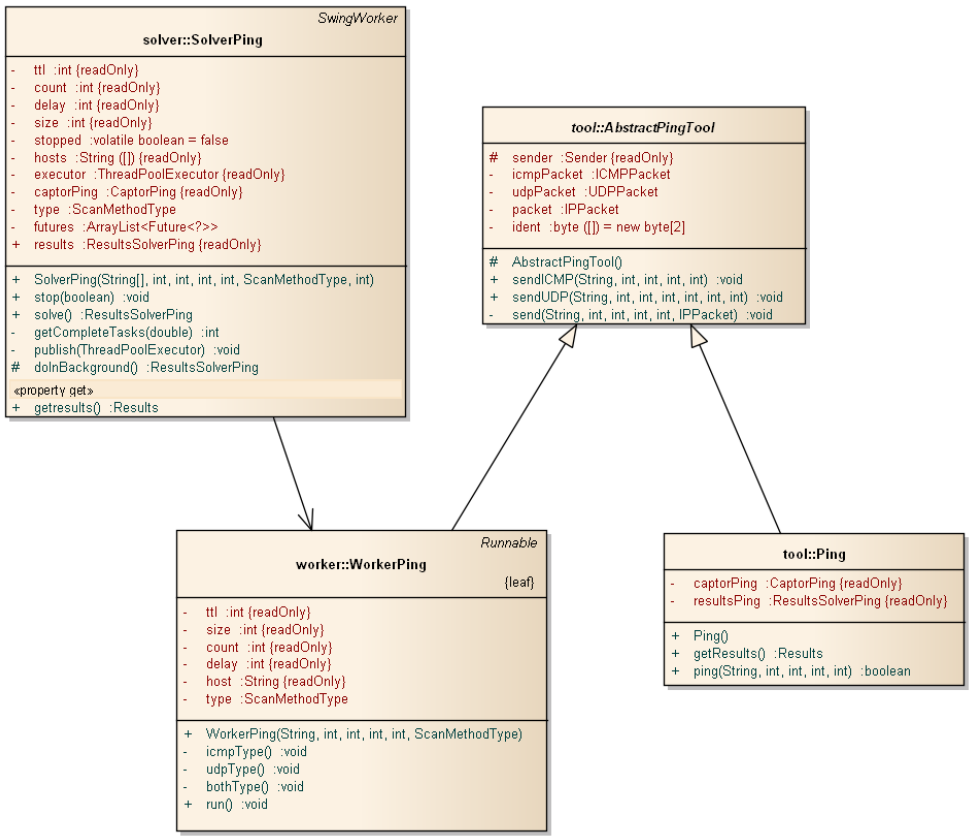
[29] Event Dispatcher Thread – vlákno řešící správu události v GUI komponentách.

Jako výstup tohoto procesu je navrácena sada dostupných hostů zastoupena třídou **ResultsSolverPing**, která zapouzdřuje kolekci typu *HashMap* a obaluje ji potřebnou logikou pro práci se sadou výsledků.

Logika odesílání je reprezentována abstraktní třídou **AbstractPingTool**, která slouží jako potenciální předek pro případné specializace, ty jsou v aplikaci dvě:

- WorkerPing – třída implementující rozhraní *Runnable* uzpůsobená pro běh ve vláknech.
- Ping – třída sloužící jako jednoduchá implementace nástroje *ping* pomocí níž je možné ověřit, zda je dostupný uzel. Třída byla implementována za účelem testování a jinak nemá v aplikaci zastoupení.

Odesílat je možné ICMP pakety typu 8 a UDP pakety na zvolený port.



Obrázek 52 – třídy nástroje řešícího detekci uzlů

Samotná třída *SolverPing* plní úlohu dozorčího vlákna a při zahájení činnosti nejprve nastartuje zachytávající vlákno, posléze vytvoří fond vláken a naplní frontu úloh. O řešení jednotlivých úloh se starají pak dělnická vlákna reprezentována třídou **WorkerPing**. O bezpečný princip návrhového vzoru se stará zapouzdřená instance třídy **ThreadPoolExecutor** z balíčku *java.util.conncurent* přímo z JDK, která řeší konkurenční přístup k frontě a přidělování úloh vláknům.

Během vývoje docházelo k občasným problémům, které byly způsobeny externími programy, jež generovaly obdobné druhy paketů jako vyvíjená aplikace.

Konkrétně docházelo k zachycení paketů, jež aplikace sama nevygenerovala. Bylo tedy nutné nějakým způsobem rozlišit identitu paketů vytvořených aplikací samotnou od těch vzniklých mimo ní. K tomuto účelu každý odeslaný paket obsahuje identifikátor aplikace a konkrétního nástroje. Identifikátor byl uložen pro:

- ICMP paket v poli **identifier**,
- UDP paket v poli **identification** v IP záhlaví z důvodu diskutovaného výše.

Sekvenční číslo nebylo pro potřeby detekce dostupnosti využito. Pomocí **identifikátoru** a **IP adresy** cílového uzlu bylo již možné pakety organizovat dle potřeb.

Tento nástroj umožňuje odpovědět na otázku, zda je daný uzel dostupný, či nikoliv a v případě, že ano, kolik paketů bylo odesláno a přijato.

5.5.3 Detekce tras

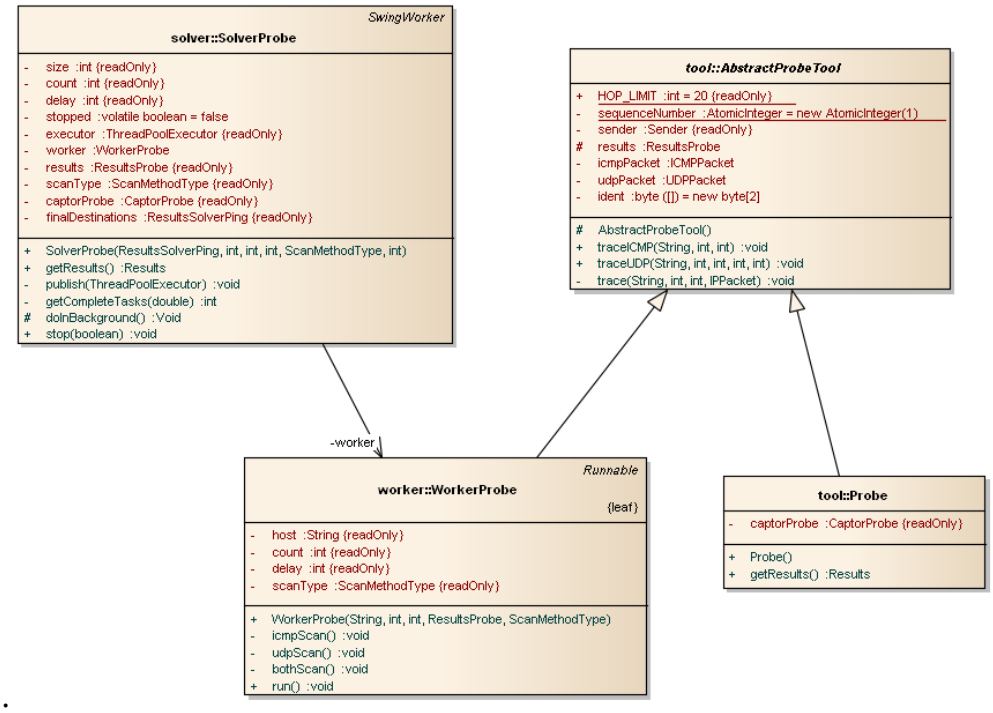
O detekci tras se stará nástroj reprezentovaný třídou **SolverProbe**, která stejně jako předešlý nástroj pro detekci uzlů rozšiřuje třídu *SwingWorker* z důvodů diskutovaných v odstavci Detekce uzlů a rovněž plní úlohu dozorčího vlákna, které řídí celý proces detekce. Zprvu nastartuje zachytávající vlákno, poté vytvoří fond vláken a naplní frontu úloh.

Činnost třídy je závislá na výstupu dříve představeného nástroje pro detekci uzlů a tedy jako své parametry přebírá sadu **ResultsSolverPing** a seznam parametrů ovlivňujících průběh detekce:

- velikost odesílaného paketu,
- počet odesílaných paketů na jeden uzel,
- zpoždění mezi odesíláním jednotlivých paketů na konkrétní uzel,
- typ použité metody UDP, ICMP nebo obojí,
- počet vláken, jež budou řešit úlohy.

Logika odesílaní je reprezentována abstraktní třídou **AbstractProbeTool**, která slouží jako předek pro případné specializace, jež jsou v aplikaci dvě:

- WorkerProbe – verze pro použití řešení úloh ve vláknech,
- Probe – jednoduchá implementace nástroj *traceroute*, sloužící pro testovací účely, jinak nemá v aplikaci žádné zastoupení.



Obrázek 53 – třídy nástroje řešícího detekci tras

Systém celé detekce je poněkud komplikovanější, než tomu bylo u kontroly pouhé dostupnosti. Funguje na principu odesílání paketů s narůstající hodnotou TTL a ukládání reakcí směrovačů na tyto pakety. Jak bylo poznamenáno v teoretické části, směrovač je dle specifikace povinen snížit hodnotu TTL minimálně o jedna a v případě snížení na hodnotu nula dojde k informování odesílající strany o této skutečnosti.

Protože navrátivší pakety mohou přicházet v různém pořadí, případně se nevrátí vůbec, bylo nutné zajistit jejich korektní identifikaci, která se využije při sestavování trasy, po které pakety putovaly. Proto každý paket obsahuje:

- **identifikátor nástroje** – který paket generuje z důvodu rozlišení jednotlivých nástrojů,
- **IP adresu** – identifikátor cílové stanice, na kterou míří (mj. povinná položka),
- **sekvenční číslo** – určující hodnotu TTL a tedy i pořadí odesílání.

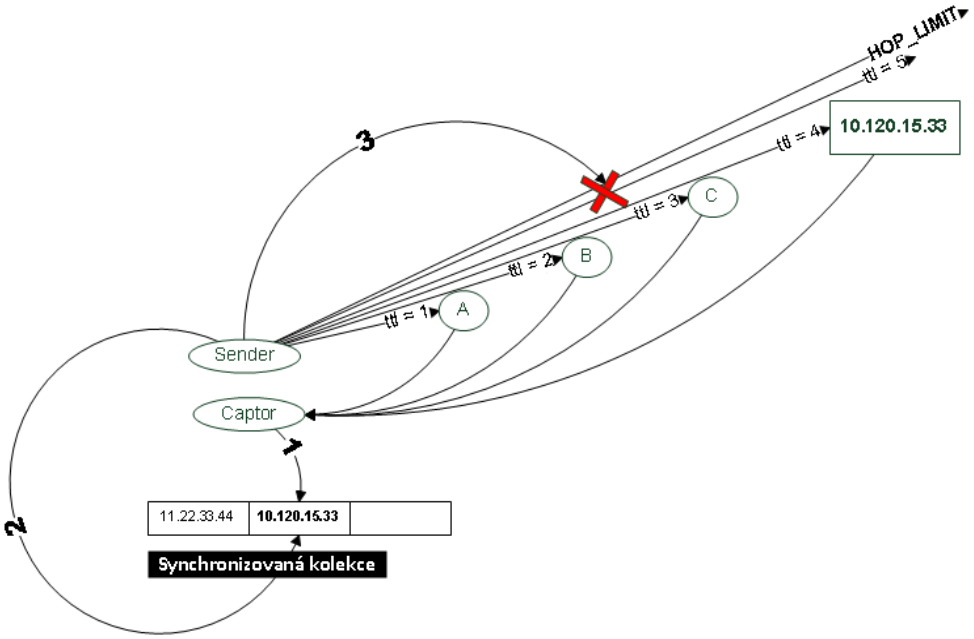
Odesílat je možné ICMP pakety nebo UDP, záleží na volbě uživatele třídy. Pro odeslané pakety ICMP typu 8 je pro položky identifikátor a sekvenční číslo využito odpovídajících polí přímo v ICMP záhlaví. Pro pakety UDP jsou položky uloženy jako kombinace bytů do IP záhlaví pole identification (z důvodů diskutovaných výše v textu), odkud jsou pak při inspekci paketu vyseparovány.

Každý detekční proces disponuje limitní hodnotou, po jejímž překročení je trasovaná stanice prohlášena za nedostupnou. Hodnota je označena jako *HOP_LIMIT* (značí maximální možnou hodnotu TTL) a je definována jako konstanta ve třídě starající se o odesílání *Abs-*

tractProbeTool. Postupně tedy dochází ke generování paketů s hodnotami 1, 2, 3 až do hodnoty *HOP_LIMIT*.

Ovšem paket může dosáhnout cíle dříve, než je hodnota *HOP_LIMIT* a v takovém případě by zbytečně detekce plýtvala systémové prostředky, proto bylo nutné vytvořit mechanismus, který zajistí signalizaci, že bylo dosaženo cíle. Vzhledem k tomu, že odesílaní a zachytávaní probíhalo v různých vláknech, bylo třeba jejich součinnost synchronizovat.

K tomuto účelu byla použita synchronizovaná kolekce, do které zachytávající vlákno v případě dosažení cíle, uložilo identifikátor uzlu a na straně odesílajícího vlákna se tato hodnota ověřila. Pokud se v kolekci cílových hostů již záznam nacházel, došlo k ukončení odesílajícího vlákna.



Obrázek 54 – princip ukončování odesílajících vláken při detekci tras

Výstupem tohoto procesu je instance třídy **ResultsProbe**, která zapouzdřuje naměřená data. Pro každou sledovanou destinaci jsou příchozí pakety přidruženy ke konkrétní trase identifikované IP adresou cíle. Pakety jsou organizovány v pořadí, ve kterém dorazily (to ale nemusí korelovat s pořadím odeslání).

5.5.4 Watchdog

Je proces, který běží na pozadí modelu a kontroluje, zda jsou již zobrazené uzly aktivní a v případě, že tomu tak není, signalizuje tento stav vizuálně. Pro dostupné uzly je typická zelená barva, naproti tomu nedostupné uzly charakterizuje barva červená. De facto se jedná o obdobu detekce uzlů popsané v předešlém textu s tím rozdílem, že nepřebírá uživatelem zadaný rozsah, ale čte data přímo z modelu.

5.6 Vizualizace

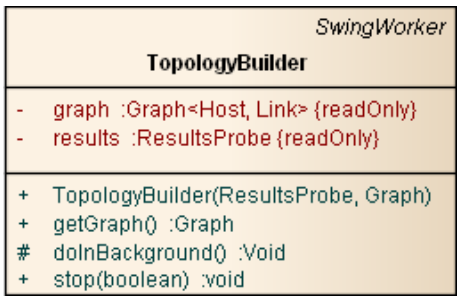
Data získaná v detekční fázi bylo nutné nejprve normalizovat, aby odpovídala skutečnosti. Jak bylo poznamenáno dříve, u jednotlivých tras se uchovávaly pakety v pořadí, v jakém dorazily, což ale nemusí korespondovat s pořadím odeslaní, protože může dojít na cestě ke zpoždění (např. paket putuje jinou cestou), případně paket nemusí vůbec dorazit. Při zobrazení neupravených dat by tak model sítě závisel na náhodě, což není žádoucí jev. Snaha byla o co nejpřesnější možné zobrazení reality.

5.6.1 Budování topologie

Transformaci dat získaných detekcí a naplnění datové struktury graf (třída *DirectedSparseGraph* diskutovaná dříve v textu) má v kompetenci třída **TopologyBuilder** rozšiřující třídu *SwingWorker*. Třída v samostatném vlákně provede:

1. normalizaci trasy,
2. zanesení hostů a propojovacích linek do grafové struktury.

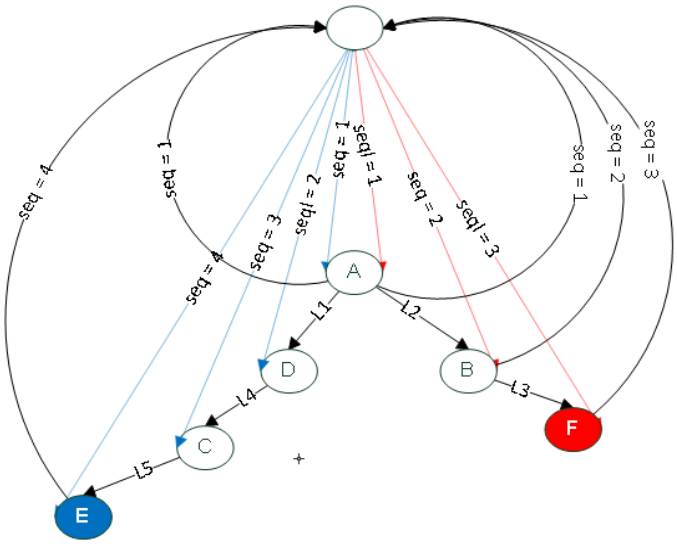
O průběhu výpočtu je uživatel informován pomocí progres barů a po dokončení je na výstupu k dispozici připravená topologická struktura.



Obrázek 55 – třída TopologyBuilder

Normalizaci zajišťuje pomocná třída **Trace** a její metoda **normalizePath()**, která provede:

1. setřídění dat podle sekvenčního čísla,
2. označení typů jednotlivých uzlů na základě jejich pořadí,
   - první na trase,
   - prostředník,
   - poslední na trase,
3. detekci chybějících uzlů na základě narušení řady po sobě jdoucí sekvenčních čísel. V takovém případě bude počet chybějících uzlů zaznamenán a uložen spolu s identifikátorem trasy k lince, na které byl problém zjištěn.

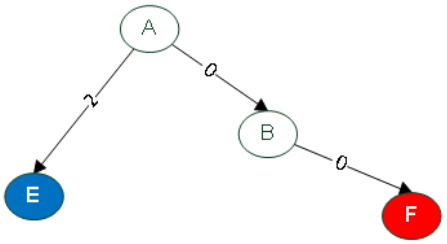


Obrázek 56 – fiktivní detekce tras

Tabulka 3 – přehled detekce

Trasa E		**Trasa F**	
Linka	**Chybí**	**Linka**	**Chybí**
AE	2	AB	0
		BF	0

Z obrázku je patrné, že při detekci trasy pro cílovou stanici **E**, neodpověděly směrovače **C** a **D** ze sekvenčních čísel, jež se navrátily, nelze odvodit konkrétní uzly, ale je možné zjistit jejich počet. V tomto případě se tedy jedná o dva. Na výstupu bude zobrazena trasa k cíli E pouze mezi A a E s hodnotou chybějících uzlů 2, která se zobrazí na základě uživatelské interakce.



Obrázek 57 – vykreslená mapa zjištěná předešlou detekcí

Po normalizaci jsou data zanesena do grafové struktury a ta je předána do komponenty starající se o zobrazení.

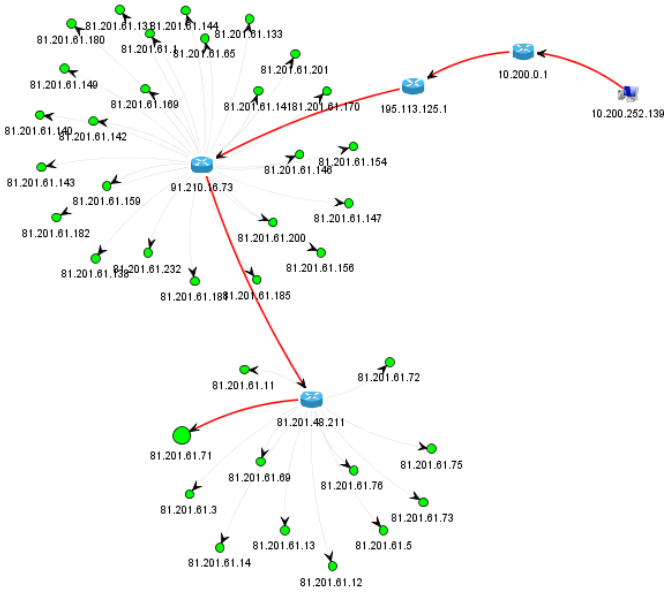
5.6.2 Vizualizace grafové struktury

Komponenta starající se o zobrazení je reflektována třídou **MainWindow**, ve které se topologickým datům přiřadí 2D koordináty na základě vhodného algoritmu rozložení.

Jako základní algoritmus rozložení byl zvolen FRLayout, tedy algoritmus řízený silou implementace Fruchterman-Reinghold.

Tento konkrétní algoritmus byl zvolen na základě série testů dostupných algoritmů z knihovny JUNG2. Při testování docházelo ke sledování vlivu změny parametrů na výsledný grafický výstup u různých algoritmů. A vybraný kandidát poskytoval nejlepší estetický vzhled (hodnoceno subjektivně).

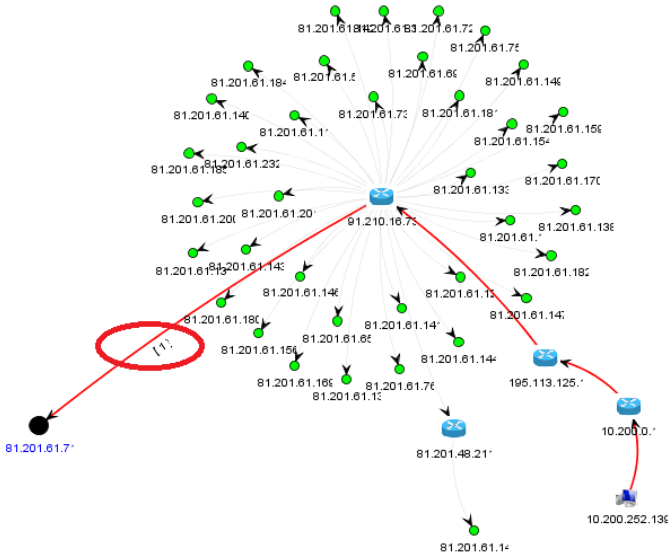
Nicméně aplikace dává možnost po vykreslení pomocí předvoleného algoritmu zvolit rozložení jiné. Tato funkcionalita byla doplněna z důvodu testování a podklady z ní jsou použity v teoretické části jako demonstrace jednotlivých rozložení.



Obrázek 58 – vykreslení menší topologie a vyznačení trasy k uživatelem zvolenému cíli

5.6.3 Paralelismus a problémy

V průběhu vývoje byly odhaleny některé problémy, které způsobovaly nepřesnosti v modelu, respektive byla prokázána jistá korelace mezi počtem vláken řešících detekci trasy a počtem chybějících uzlů v některých úsecích.



Obrázek 59 – dopad počtu vláken na chybějící směrovače

Na obrázku výše je zobrazena tatáž topologie jako na Obrázek 58 – vykreslení menší topologie a vyznačení trasy k uživatelem zvolenému cíli. Na první pohled je zřejmé, že vypadá

odlišně (ve skutečnosti měly být některé uzly navázané na směrovač *91.210.16.73* navázané až na *81.201.48.211*). Není pravděpodobné, že by se fyzická struktura změnila tak pohotově (interval mezi testováním nebyl nikterak velký, řádově sekundy).

Na vyznačené trase je indikován chybějící směrovač. Jak se později ukázalo, je tato nepřesnost způsobena nastavením některých směrovačů, které paralelně příchozí dotazy určitého typu vyhodnotí jako možný útok[30] a odpovědi na ně ignorují nebo odpoví pouze na jeden. Směrovače jsou designované na co nejrychlejší obsluhu paketů a v případě zahlcení (např. přetečení bufferu) mohou selhat, případně pozměnit své fungování. Dle zjištěných informací je toto chování poměrně přirozené.

Na základě tohoto faktu, byla do GUI aplikace doplněna možnost ovlivnit počet řešících vláken dle uvážení uživatele. V některých sítích totiž nebyl s větším počtem vláken problém. Pokud chce mít uživatel ovšem jistotu, je možné detekci řešit pouze jedním vláknem za cenu snížení rychlosti detekce.

5.6.4 Uživatelská interakce

Po zobrazení topologické mapy má uživatel několik možností jak s mapou pracovat. Aplikace nabízí základní operace, jež lze u nástrojů podobného charakteru očekávat:

- manipulace s jednotlivými objekty (odstranění ani ruční vkládání není implementováno),
- dva režimy interakce a transformace,
- zoom, rotace pohyb s celou mapou,
- satelitní náhled na celou mapu usnadňující orientaci v hlavní mapě,
- základní možnost uložení a načtení mapy do/z souboru (XML soubor).

Pro usnadnění manipulace s objekty, byla implementována funkcionalita, která umožňuje uživateli při kliku na směrovač označit zároveň i všechny jeho následníky (cílové stanice) a manipulovat tak s celou sadou. Obvykle je snaha manipulovat s přilehlými uzly navazujícími na směrovač jako s celkem. Vyjmout uzel ze skupiny lze pomocí stisku klávesy **shift** a kliku myší na prvek, jež má být z výběru odstraněn.

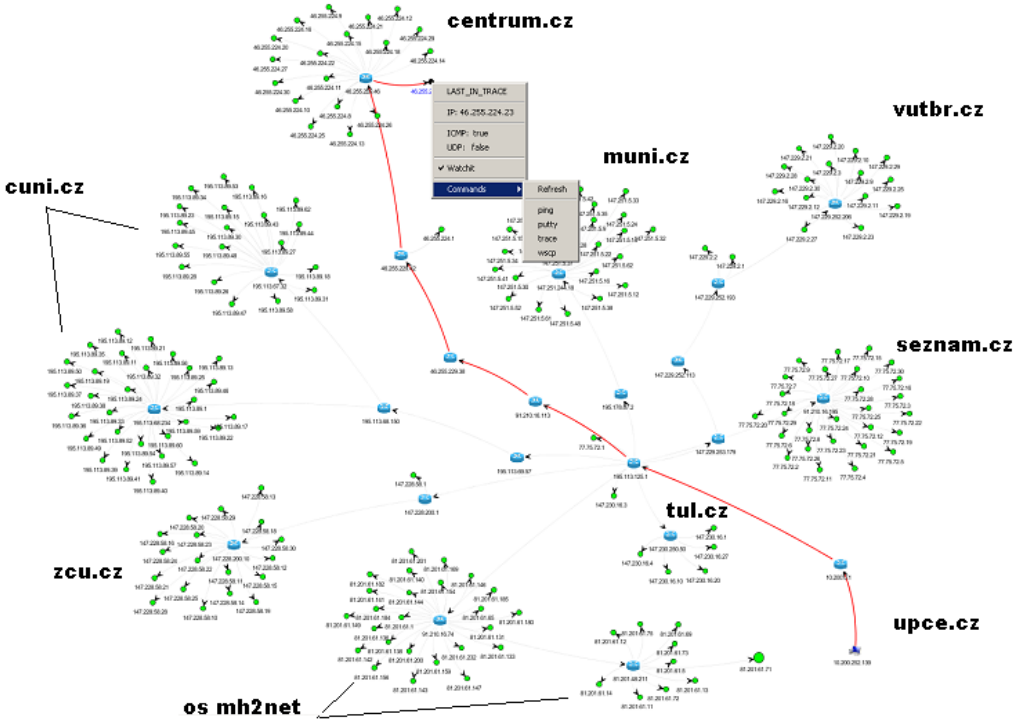
Dále má uživatel možnost ovlivňovat pomocí dialogového okna parametry skenovaní u jednotlivých detekčních nástrojů a tím tak ovlivnit kvalitu výsledného modelu. K dispozici je několik předdefinovaných schémat nastavení:

- quick – rychlé testovací schéma řešící úlohy všemi vlákny,
- spy – pomalý režim snažící se nevybočovat svým chováním od běžných nástrojů,
- DoS – agresivní schéma vhodné spíše pro testovací účely,
- single – režim, při kterém se všechny úlohy řeší jedním vláknem,
- uživatelské – uživatelem navolené parametry.

[30] TTL Expiry Attack Identification and Mitigation [50].

Při opakované detekci je možnost vynechat krok detekce hostů a pomocí uživatelské volby provádět konstrukci tras na již detekovaných uzlech (pokud jsou k dispozici, první detekci nelze vynechat), čímž se může urychlit průběh opakované detekce.

Po každé detekci je zároveň možné spojit předešlou sadu výsledků s nově detekovanými uzly a tím tak vybudovat komplexnější model sestavený z více rozsahů.



Obrázek 60 – ukázka spojování sad výsledků

Jak je patrné z obrázku, lze prostřednictvím uživatelského podnětu zobrazit trasu, u které je názorné pořadí směrovačů na trase a případně zobrazeny počty chybějících kusů. To umožní uživateli se lépe orientovat v topologii.

Z obrázku je rovněž patrně, že aplikace dovoluje pro každý uzel nadefinovat sadu externích příkazů, které lze při práci s modelem vyvolávat pro daný uzel a tím tak zefektivnit práci při odhalování potíží se sítí.

6 Závěr

V úvodu diplomové práce byla představena množina dostupných softwarových řešení, která umožňují zmapovat topologii počítačové sítě a následně vytvořit její grafickou podobu (model). Jednotlivé dostupné nástroje jsou členěny na základě velikosti rozsahu sítě, pro niž se jejich nasazení nejlépe hodí.

Dále se teoretická část zabývá popisem metodik použitelných v sítích IPv4 (více méně ty, nad nimiž pracují softwarové produkty popsané v úvodu práce), jež je možné využít při detekci topologie. Tyto možnosti jsou v prvé řadě rozděleny na aktivní a pasivní způsoby, přičemž pasivním metodám je věnována detailnější diskuze, protože se ze své podstaty více hodí při realizaci praktické části diplomové práce. Hlavní důvody pro výběr jsou vysvětleny přímo v odpovídající kapitole věnované pasivním metodikám.

V závěru teoretické části je věnována pozornost způsobům zobrazení grafové struktury v euklidovském 2D prostoru. Shrnuty jsou obecné problémy, které tato oblast přináší, zejména však přiřazení vhodných koordinátů objektům, jež nenesou žádné informace o své poloze v prostoru. Kapitola je doplněna o ilustrace, které zachycují výstupy některých algoritmů vhodných pro estetické zobrazení grafové struktury.

Praktická část se věnuje návrhu vlastního nástroje, který mapuje topologii počítačové sítě a vhodným způsobem nabízí její zobrazení doplněné o uživatelskou interakci, která umožní s výsledným modelem pracovat.

V průběhu vývoje se vyskytlo několik problémů týkajících se chybně implementovaných funkcionalit open sourcových knihoven, které byly v projektu využity. Navíc bylo zjištěno, že některé implementace protokolů TCP/IP nedodržují doporučení uvedené ve specifikaci RFC. Nicméně se podařilo veškeré zjištěné nedostatky opravit a zajistit tak korektní fungovaní výsledné aplikace.

Hlavním přínosem výsledné aplikace je snaha odstranit nedostatky dostupných řešení a poskytnout tak uživateli jednoduchý, výkonný a levný způsob vhodný při počáteční orientaci v neznámé síťové infrastruktuře. Výsledný produkt, tak splňuje veškeré cíle definované v úvodu práce.

Po dokončení aplikace bylo zjištěno, že pro její širší uplatnění je vhodné ji v budoucnu rozšířit (nad rámec zadání) o další funkcionality, které z ní učiní ještě univerzálnější nástroj. Primárně je vhodné doplnit funkcionalitu jako je možnost reakce na změny v topologii (nyní je aplikace schopná dostatečně modelovat pouze sítě se statickým směrováním). Dále pak doplnit podporu pro export výsledné mapy do grafické podoby ve formě obrázku a případně za zvážení stojí i rozšíření o uživatelskou volbu různých datových úložišť (aktuální verze podporuje pouze XML formát).

Literatura

[1] Advanced network management solution. *IBM Software.* [Online] [Citace: 20. duben 2013.] http://www-01.ibm.com/software/tivoli/products/netview/.

[2] **Šrámek, Petr.** Archív diplomových prací ČVUT. *Správa virtuálních lokálních sítí.* [Online] 2007. [Citace: 18. duben 2013.] https://dip.felk.cvut.cz/browse/details.php?f=F3&d=K13136&y=2007&a=sramep1&t=dipl.

[3] IBM Tivoli Netview . *Gambit Communications.* [Online] [Citace: 17. duben 2013.] http://www.gambitcomm.com/site/products/mgmt_apps/interopnet98.htm.

[4] Tivoli Network Manager IP Edition. *IBM Software.* [Online] [Citace: 16. duben 2013.] http://www-03.ibm.com/software/products/us/en/ibmtivolinetworkmanageripedition/.

[5] IBM Tivoli Network Manager IP Edition. *Computel system management.* [Online] [Citace: 17. duben 2013.]

[6] HP Network Node Manager i software. *HP Network Management.* [Online] HP. [Citace: 18. duben 2013.] http://www8.hp.com/us/en/software-solutions/software.html?compURI=1170657#.UYRkFEp8P1C.

[7] HP OpenView Network Node Manager. *Salto spol. s r.o.* [Online] Salto, spol. s r. o., 2013. [Citace: 20. duben 2013.] http://www.salto.cz/produkty/app/hp_openview_nnm.php.

[8] **Hussain, Nazri.** HP OpenView Network Node Manager 7.51. *Nazri's FotoPage.* [Online] 2007. [Citace: 20. duben 2013.]

[9] SNMP software třetích stran . *HW group.* [Online] [Citace: 20. duben 2013.] http://www.hw-group.com/software/pd_snmp_cz.html#CA.

[10] Nova verze CA Unicenter NSM. *Lupa.* [Online] 19. září 2006. [Citace: 20. duben 2013.]

[11] New network for MIMIC 12.00 . *Gambit communications.* [Online] 10. říjen 2012. [Citace: 20. duben 2013.] http://gambitcomm.blogspot.cz/2012/04/new-network-for-mimic-1200.htmls.

[12] Solstice Enterprise Manager 4.1 Developing Java Applications. *Using the Java Topology API.* [Online] [Citace: 2. květen 2013.] http://docs.oracle.com/cd/E19092-01/sol.entmgr41/816-2004/ch_04_topology.html#5529.

[13] OpenNMS. *Wikipedia, the free encyclopedia.* [Online] 8. duben 2013. [Citace: 20. duben 2013.] http://en.wikipedia.org/wiki/OpenNMS#History.

[14] **OpenNMS.** About OpenNMS. *The OpenNMS Project.* [Online] [Citace: 15. duben 2013.] http://www.opennms.org/about/.

[15] Map Availability in OpenNMS . *The OpenNMS Project.* [Online] [Citace: 16. duben 2013.] http://www.opennms.org/wiki/Maps.

[16] A good network monitoring solution, and how this mess happened. *Seeing Binary.* [Online] 6. listopad 2011. http://www.seeingbinary.com/2011/11/good-network-monitoring-solution-and.html.

[17] How Nagios Compares To OpenNMS. *Nagios is the industry-standard in IT infrastructure monitoring.* [Online] září 2011. [Citace: 21. duben 2013.] http://assets.nagios.com/datasheets/compare/How_Nagios_Compares_To_OpenNMS.pdf.

[18] Comparison with other network management systems. *The OpenNMS Project.* [Online] 28. červenec 2010. [Citace: 22. duben 2013.] http://www.opennms.org/wiki/Comparison_with_other_network_management_systems.

[19] Zenoss Interface and Navigation. *IT Organizations use Zenoss server, network, and cloud monitoring to manage their dynamic datacenters.* [Online] [Citace: 22. duben 2013.] Zenoss Interface and Navigation.

[20] Zenoss. *Wikipedia, the free encyclopedia.* [Online] 9. duben 2013 . [Citace: 23. duben 2013.] http://en.wikipedia.org/wiki/Zenoss.

[21] Getting Started with SolarWinds Network Topology Mapper. *facebook.* [Online] 14. březen 2013. [Citace: 20. duben 2013.] http://external.ak.fbcdn.net/safe_image.php?d=AQAcHBkZA7HN6dJP&url=http%3A%2F%2Fi2.ytimg.com%2Fvi%2FYIv4S8Trxro%2Fmaxresdefault.jpg&jq=100.

[22] Network Topology Mapper. *Solarwinds Unexpected Simplicity.* [Online] [Citace: 3. květen 2013.]

[23] Network Mapper and Monitor - LANState. *10-strike Software.* [Online] [Citace: 14. duben 2013.] http://www.10-strike.com/lanstate/.

[24] 10-Strike LANState 6.5. *FileCluster.* [Online] 19. únor 2013. [Citace: 23. duben 2013.] http://www.filecluster.com/Network-Tools/Network-Monitoring/Download-10-Strike-LANState.html.

[25] Zenmap GUI. *nmap.* [Online] [Citace: 19. duben 2013.] http://nmap.org/zenmap/.

[26] Nmap 6 Released. *nmap.* [Online] 21. květen 2012. [Citace: 18. duben 2013.] http://nmap.org/6/.

[27] Link Layer Topology Discovery. *Wikipedia, the free encyclopedia.* [Online] 28. únor 2013. [Citace: 23. duben 2013.] http://en.wikipedia.org/wiki/Link_Layer_Topology_Discovery.

[28] NetworkView Features. *NetworkView* . [Online] [Citace: 25. duben 2013.] http://www.networkview.com/html/features.html.

[29] Cheops-ng. *Cheops-ng.* [Online] [Citace: 25. duben 2013.] http://cheops-ng.sourceforge.net/index.php.

[30] Cheops-ng Screenshots. *Cheops-ng.* [Online] http://cheops-ng.sourceforge.net/screenshots.php.

[31] LanTopolog key features. *LanTopolog* . [Online] [Citace: 23. duben 2013.] http://www.lantopolog.com/.

[32] **Pužmanová, Rita.** *Moderní komunikační sítě od A do Z.* Praha : Computer Press, 1998. ISBN 80-7226-098-7.

[33] Force-directed graph drawing. *Wikipedia, the free encyclopedia.* [Online] 5. květen 2013. [Citace: 6. květen 2013.] http://en.wikipedia.org/wiki/Force-directed_graph_drawing.

[34] Graph drawing. *Wikipedia, the free encyclopedia.* [Online] 3. duben 2013. [Citace: 15. duben 2013.] http://en.wikipedia.org/wiki/Graph_drawing.

[35] **Weichel, Christian.** Spectral graph drawing. *32leaves.* [Online] +. červenec 2009. [Citace: 20. duben 2013.] http://32leav.es/?p=557#.

[36] **yWorks.** Automatic Graph Layout. *yFiles for Java Developer's Guide.* [Online] 2012. [Citace: 20. duben 2013.] http://docs.yworks.com/yfiles/doc/developers-guide/orthogonal_layouter.html.

[37] **MB Technologies.** Graph Layout Algorithms. *The Ajax Framework for Data Visualization.* [Online] 14. leden 2013. [Citace: 20. leden 2013.] http://www.infiview.com/dev/kb/Layouts.html.

[38] Arc diagram. *Wikipedia, the free encyclopedia.* [Online] 5. duben 2013. [Citace: 20. duben 2013.] http://en.wikipedia.org/wiki/Linear_embedding.

[39] Layered graph drawing. *Wikipedia, the free encyclopedia.* [Online] 22. březen 2013. [Citace: 17. duben 2013.] http://en.wikipedia.org/wiki/Layered_graph_drawing.

[40] **Nikos Drakos, Ross Moore.** Layered layout. *MONASH University, Information Technology, Clayton School of I.T.* [Online] 7. listopad 2006. [Citace: 21. duben 2013.] http://www.csse.monash.edu.au/hons/se-projects/2006/Kieran.Simpson/output/html/node7.html.

[41] **Meyer, Bernd.** Self-Organizing Graphs and ISOM Layout . [Online] 2. srpen 1998. [Citace: 1. duben 2013.] http://www.csse.monash.edu.au/~berndm/ISOM/.

[42] *jpcap -- a network packet capture library.* [Online] [Citace: 4. březen 2013.] http://jpcap.sourceforge.net/.

[43] Wrong id when use jpcap catching ICMP packet. *Google groups.* [Online] 18. květen 2008. [Citace: 5. březen 2013.]

[44] Incorrect fields on captured ICMP packet. *Google groups.* [Online] 5. červen 2007. [Citace: 5. březen 2013.] https://groups.google.com/forum/?fromgroups=#!topic/jpcap/IFjGbm2HFs0.

[45] **Menčík, Vlastimil.** Java 8 - Je tu konečně revoluce? *Et netera.* [Online] 28. březen 2013. [Citace: 29. duben 2013.] http://dev.etnetera.cz/cz/java/java_8_je_tu_konecne_revoluce.html.

[46] **Surguy, Ingo.** Creating an "Absolute Singleton". *Inigo Surguy.* [Online] [Citace: 20. duben 2013.] http://surguy.net/articles/communication-across-classloaders.xml.

[47] **Postel, J.** INTERNET CONTROL MESSAGE PROTOCOL. *IETF Documents.* [Online] 1981. [Citace: 19. duben 2013.] http://tools.ietf.org/html/rfc792.

[48] **M. West, S. McCann.** TCP/IP Field Behavior. *The Internet Engineering Task Force (IETF).* [Online] 2006. [Citace: 19. duben 2013.] http://www.ietf.org/rfc/rfc4413.txt.

[49] **O'Madadhain, Joshua.** Comparing open source java graph drawing frameworks(JUNG and Prefuse) for drawing network topology. *Stack Overflow.* [Online] 1. leden 2011. [Citace: 29. duben 2013.] http://stackoverflow.com/questions/4670846/comparing-open-source-java-graph-drawing-frameworksjung-and-prefuse-for-drawin.

[50] TTL Expiry Attack Identification and Mitigation. *Cisco Security.* [Online] [Citace: 20. duben 2013.]

[51] History of Zenoss Core. *Zenoss Open Source IT Management.* [Online] [Citace: 22. duben 2013.] http://community.zenoss.org/community/about.

Příloha A – zdrojový kód pro vytvoření grafu Prefuse

```
public class PrefuseDemo {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Demonstrace vizualizace grafu knihovna
Prefuse.");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final Graph graph = new Graph();
        setupGraph(graph);
        final ColorAction vertexies = new ColorAction("graph.nodes", Vi-
sualItem.FILLCOLOR, ColorLib.rgb(255, 0, 0));
        final ColorAction edges = new ColorAction("graph.edges", VisualI-
tem.STROKECOLOR, ColorLib.gray(200));
        final ActionList color = new ActionList() {{add(vertexies);
add(edges); }};
        final ActionList layout = new ActionList() {{add(new Rando-
mLayout("graph"));add(new RepaintAction());}};
        final ShapeRenderer r = new ShapeRenderer();
        Visualization vv = new Visualization() {{add("graph", graph);
putAction("color", color); putAction("layout",
layout);setRendererFactory(new DefaultRendererFactory(r));}};
        frame.add(new Display(vv){{setSize(new Dimension(640,480));}});
        frame.pack();
        frame.setVisible(true);
        vv.run("color");
        vv.run("layout");
    }

    public static void setupGraph(Graph graph) {
        Random r = new Random();

        for (int i = 0; i < 20; i++) {
            graph.addNode();
        }
        for (int i = 0; i < 30; i++) {
            Integer v1 = r.nextInt(20);
            Integer v2 = r.nextInt(20);
            graph.addEdge(v1, v2);
        }
    }
}
```

Příloha B – zdrojový kód pro vytvoření grafu JUNG2

```
public class JUNG2Demo {

    public static void main(String[] args) {
        JFrame frame = new JFrame("Demonstrace vizualizace grafu knihovna
JUNG2.");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        DirectedSparseMultigraph<Integer, String> graph = new Directed-
SparseMultigraph<>();
        setupGraph(graph);
        FRLayout<Integer,String> layout = new FRLayout<>(graph);
        VisualizationViewer<Integer,String> vv = new Visualization-
Viewer(layout,new Dimension(640,480));
        frame.add(vv);
        frame.pack();
        frame.setVisible(true);
    }

    public static void setupGraph(Graph graph) {
        Random r = new Random();

        for (int i = 0; i < 20; i++) {
            graph.addVertex(new Integer(i));
        }
        for (int i = 0; i < 30; i++) {
            Integer v1 = r.nextInt(20);
            Integer v2 = r.nextInt(20);
            String edge = v1.toString() + "-" + v2.toString();
            graph.addEdge(edge, v1, v2);
        }
    }
}
```

Příloha C – uživatelský návod

Příloha C slouží jako uživatelský návod a napomáhá uživateli rychleji se s aplikací seznámit a pochopit principy fungování. Víceméně se bude jednat o obrázkového průvodce s náležitým popisem klíčových polí. Ke spuštění je zapotřebí pouze:

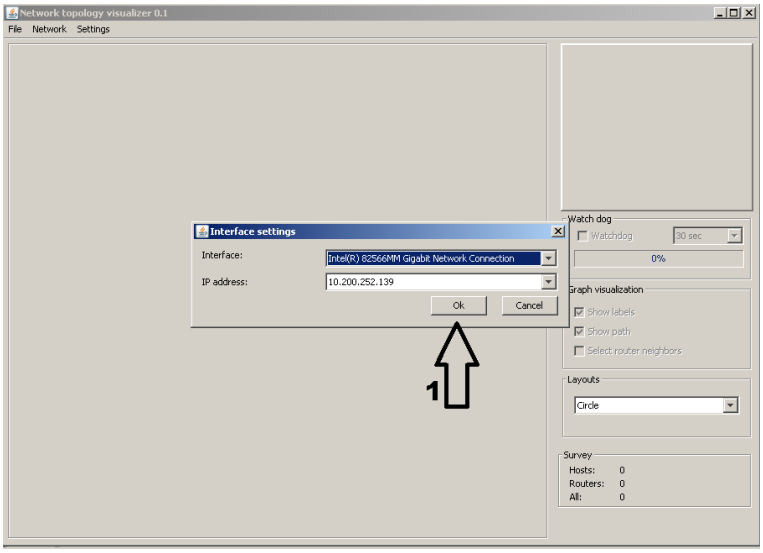
- běhové prostředí Javy (JRE testováno na verzi Java 1.7.),
- nainstalovaná knihovna libcap/winpcap (verze pro OS Windows je na přiloženém CD nosiči v adresáři ***prakticka_cast/libs/WinPcap_4_1_3.exe***).

Před samotným spuštěním je nutné knihovnu winpcap nainstalovat!

Spuštění

Aplikaci je možné spustit dvěma způsoby:

- dvojklikem myši na spouštěcí soubor *NetVisualizer.jar*,
- z konzole pomocí příkazu *java –jar NetVisualizer.jar*.



Obrázek 61 – první spuštění

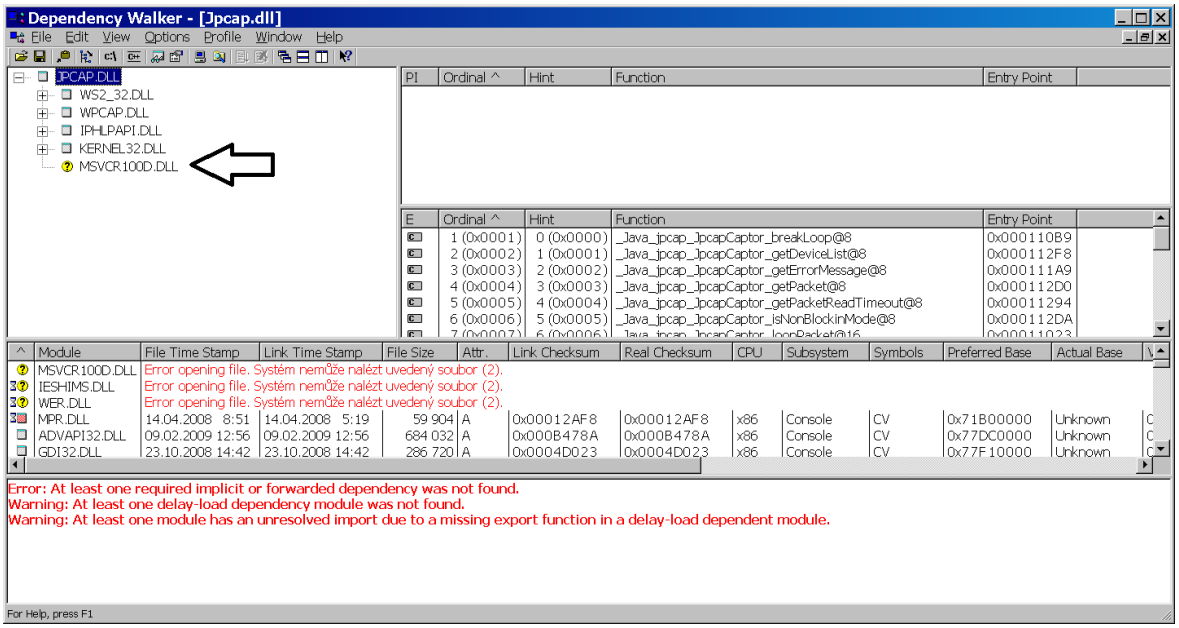
Při prvním spuštění je uživatel dotázán na výběr síťově karty a IP adresy. Pokud má síťová karta přiřazeno více IP adres, je možné zvolit tu, kterou ponesou odchozí pakety jako zdrojovou. Při dalším spuštění již uživatel není vybízen k zadání údajů, nicméně je možné síťovou kartu/IP adresu dodatečné změnit v položce menu *Settings – Interfaces.*

Aplikaci není možné spouštět z přiloženého CD (adresáře bin32 a bin64), protože potřebuje zapisovat na disk konfigurační soubory!

Problémy při spuštění

Pokud se při spuštění vyskytnou problémy, že knihovna Jpcap.dll (u Windows) nemůže nalézt navázané knihovny, jak je znázorněno v Zdrojový kód 9 – nemožnost načíst naváza-

né knihovny. Je vhodné využit nástroj pro kontrolu provázanosti knihoven jako např. **Dependency Walker** (pro platform x86 je program dostupný na přiloženém CD**)**, který odhalí případné chybějící knihovny.



Obrázek 62 – chybějící knihovna msvcr100d.dll

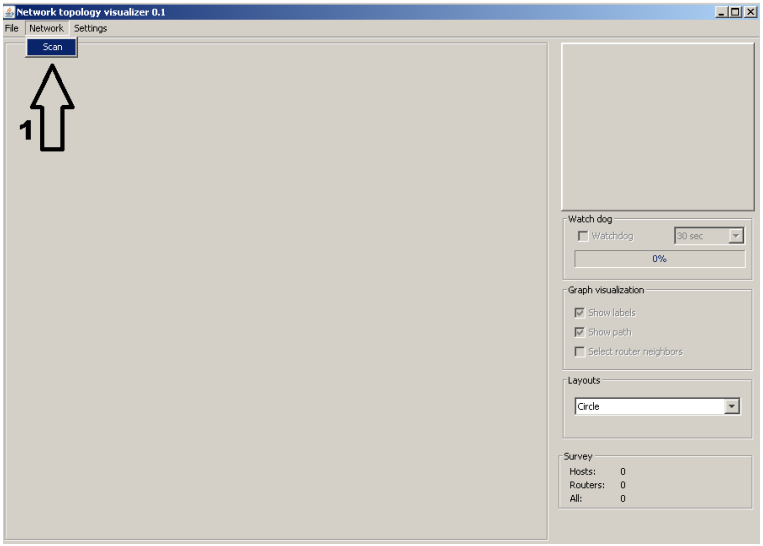
Z obrázku je patrné že chybí knihovna **msvcr100d.dll**. Po jejím doplnění již bylo možné aplikaci bez problémů spustit a používat. Popsané problémy vykazoval operační systém Windwos XP Service pack 3. Na Windows 7 fungovala aplikace korektně. Nicméně je možné, že mohou chybět i jiné knihovny. Postup pro zjednání nápravy je však totožný.

```
e:\prakticka_cast\bin32>java -jar  NetVisualizer.jar
Exception in thread "AWT-EventQueue-0" java.lang.UnsatisfiedLinkError:
C:\WINDOWS\system32\Jpcap.dll: Can't find dependent libraries
        at java.lang.ClassLoader$NativeLibrary.load(Native Method)
        at java.lang.ClassLoader.loadLibrary1(Unknown Source)
        at java.lang.ClassLoader.loadLibrary0(Unknown Source)
        at java.lang.ClassLoader.loadLibrary(Unknown Source)
        at java.lang.Runtime.loadLibrary0(Unknown Source)
        at java.lang.System.loadLibrary(Unknown Source)
        at jpcap.JpcapCaptor.<clinit>(JpcapCaptor.java:251)
        at
cz.upce.fei.netvis.net.utils.NetworkUtils.getInterfacesList(NetworkUtils.
java:86)
        at
cz.upce.fei.netvis.view.ApplicationSettingsDialogWindow.<init>(Applicatio
nSettingsDialogWindow.java:68)
        at cz.upce.fei.netvis.view.MainWindow.<init>(MainWindow.java:138)
        at cz.upce.fei.netvis.view.MainWindow$17.run(MainWindow.java:960)
        at java.awt.event.InvocationEvent.dispatch(Unknown Source)
        at java.awt.EventQueue.dispatchEventImpl(Unknown Source)
        at java.awt.EventQueue.access$200(Unknown Source)
        at java.awt.EventQueue$3.run(Unknown Source)
        at java.awt.EventQueue$3.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at ja-
va.security.ProtectionDomain$1.doIntersectionPrivilege(Unknown Source)
        at java.awt.EventQueue.dispatchEvent(Unknown Source)
        at java.awt.EventDispatchThread.pumpOneEventForFilters(Unknown
Source)
        at java.awt.EventDispatchThread.pumpEventsForFilter(Unknown Sour-
ce)
        at java.awt.EventDispatchThread.pumpEventsForHierarchy(Unknown
Source)
        at java.awt.EventDispatchThread.pumpEvents(Unknown Source)
        at java.awt.EventDispatchThread.pumpEvents(Unknown Source)
        at java.awt.EventDispatchThread.run(Unknown Source)
```

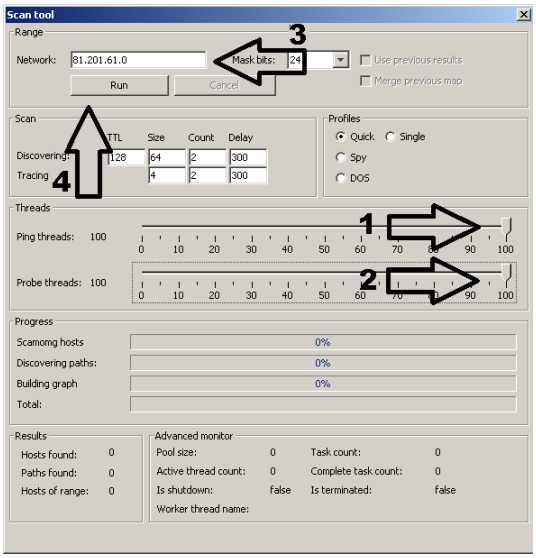
Zdrojový kód 9 – nemožnost načíst navázané knihovny

Zahájení detekce

K vyvolání dialogového okna dojde pomocí volby v menu *Network – Scan.*



Obrázek 63 – vyvolání okna pro zahájení detekce



Obrázek 64 – parametry detekce

Parametry detekce je možné ovlivňovat ručně pomocí různých voleb, které jsou rozděleny na:

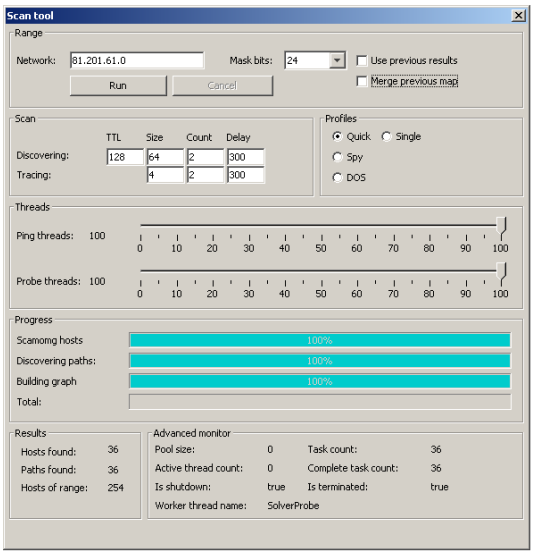
- detekce uzlů (discovering),
  - TTL – hodnota TTL odchozích paketů,
  - size – velikost datové části odchozích paketů,
  - count – počet odeslaných paketů,
  - delay – zpoždění mezi odesíláním,
- detekce tras (tracing)
  - krom hodnoty TTL se jedná o analogii k výše uvedenému. Hodnota TTL je zvyšována automaticky na základě algoritmu detekce.

Další možnost nabízí záložka **Profiles**, kde jsou výše popsané hodnoty předdefinované pomocí různých profilů:

- quick – rychlé testovací schéma řešící úlohy všemi vlákny,
- spy – pomalý režim snažící se nevybočovat svým chováním od běžných nástrojů,
- DoS – agresivní schéma vhodné spíše pro testovací účely,
- single – režim, při kterém se všechny úlohy řeší jedním vláknem.

Pomocí posuvné lišty (na obrázku vyznačeno jako **1** a **2**) lze ovlivnit počty vláken řešících danou úlohu.

Do pole vyznačeného **šipkou 3** se zadává rozsah sítě, kterou chce uživatel podrobit detekci. Tlačítkem **Run** (šipka 4) je zahájena samotná detekce.



Obrázek 65 – dokončená detekce

Po dokončení detekce má uživatel tři možnosti.

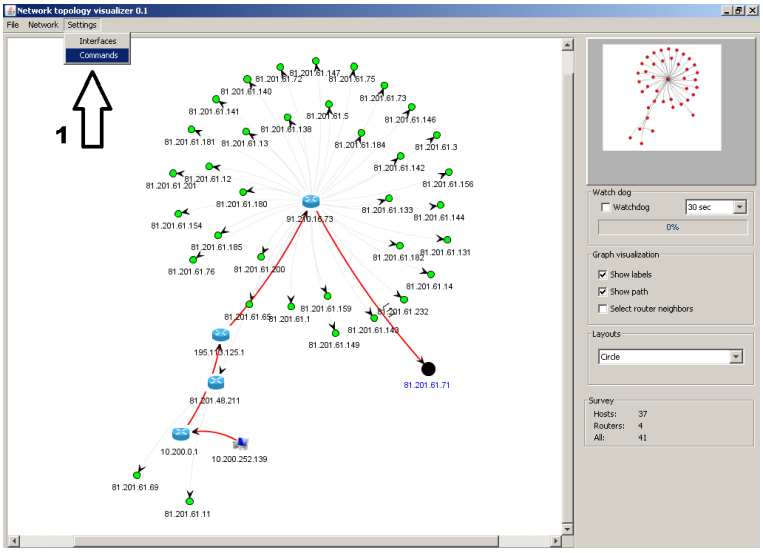
Dialogové okno zavřít – což má za následek ukončení detekce.

Use previous results – zaškrtnutím této volby a opětovným spuštěním pomocí Run (s jinými parametry detekce) dojde k **přeskočení kroku detekce uzlů** a použije se sada výsledků získaná z předešlé detekce. Tím je možné ušetřit prostředky vynaložené na detekci uzlů.

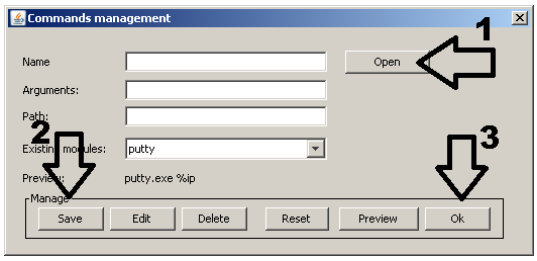
Merge previous map – pokud je tato volba aktivována, dojde k začlenění nově objevených tras k již stávajícím. Vykreslení proběhne do jedné mapy. Takto lze vytvářet komplexnější modely spojené z více různých rozsahů.

Definování uživatelských příkazů

Po dokončení detekce je zobrazen model počítačové sítě. Nad uzly je dále možné definovat sadu příkazů, které bude moci uživatel pro uzly spouštět.



Obrázek 66 – vyvolání dialogu pro zadávání příkazů

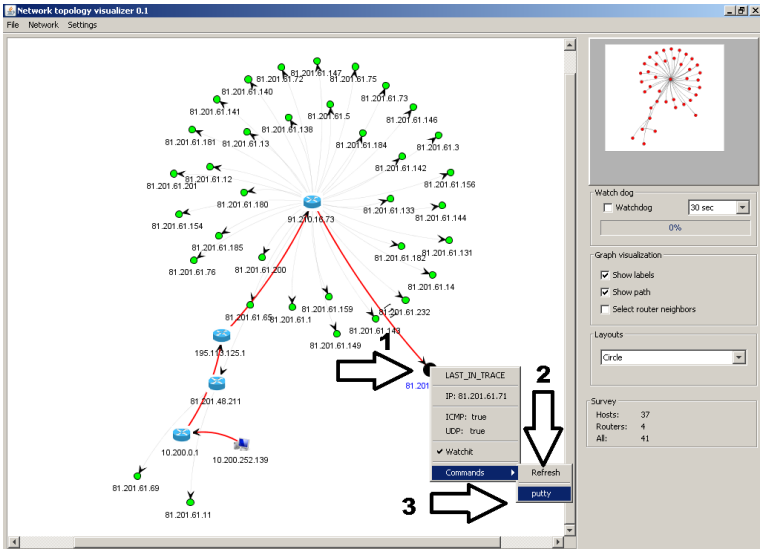


Obrázek 67 – zadávání příkazů

Každý příkaz se skládá z několika polí:

- **Name** – uživatelsky pojmenovaný příkaz.
- **Arguments** – dodatečné argumenty příkazu. V případě vložení symbolu **%IP** dojde při vykonávání příkazu k náhradě za skutečnou IP adresu uživatelem vybraného uzlu (na obrázku možno vidět v poli **preview** u již definovaného příkazu).
- **Path** – absolutní cesta ke spouštěcímu souboru daného příkazu.

Tlačítkem **Preview** je možné zobrazit náhled daného příkazu.



Obrázek 68 – vyvolání zadaného příkazu

Příkaz je možné vyvolat následujícím způsobem:

1. pravý klik myši na vybraný uzel.
2. Pokud se jednalo o nově zadaný příkaz, který není v menu vidět, je nutné kliknout na záložku **Refresh**, čímž se do menu načte (pokud se již příkaz v menu nachází, lze tento krok vynechat).
3. Vyvolání samotného příkazu nad uzlem.

Manipulace s modelem

S vykresleným modelem lze pracovat pomocí myši, klávesových zkratek a ovlivňovat pomocí voleb v nabídce. Pro korektní fungování klávesových zkratek je nutné, aby **plátno vlastnilo focus**.

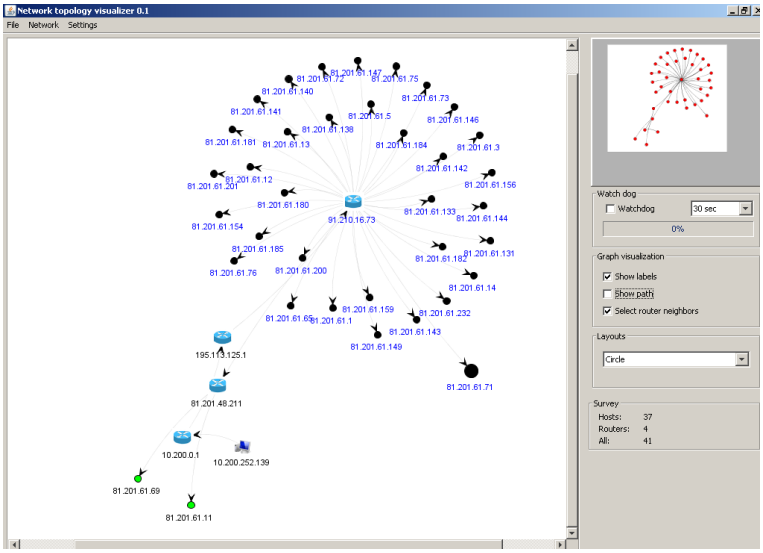
Klávesové zkratky

Klávesa **p** – režim manipulace s uzly pomocí myši:

- **výběr jednoho** – kliknutím levého tlačítka myši na uzel,
- **výběr skupiny uzlů** – tažením a stiskem levého tlačítka myši,
- **manipulace** – s uzlem stylem drag and drop,
- **odebrání ze skupiny** – pomocí levého tlačítka myši a klávesy shift,
- **zjišťování informací o uzlu** – pomocí kliku pravého tlačítka na uzel,
- **vyvolávat příkazy** – pomocí pravého tlačítka myši a výběru příkazu z menu.

Klávesa **t** – režim transformace modelu pomocí myši:

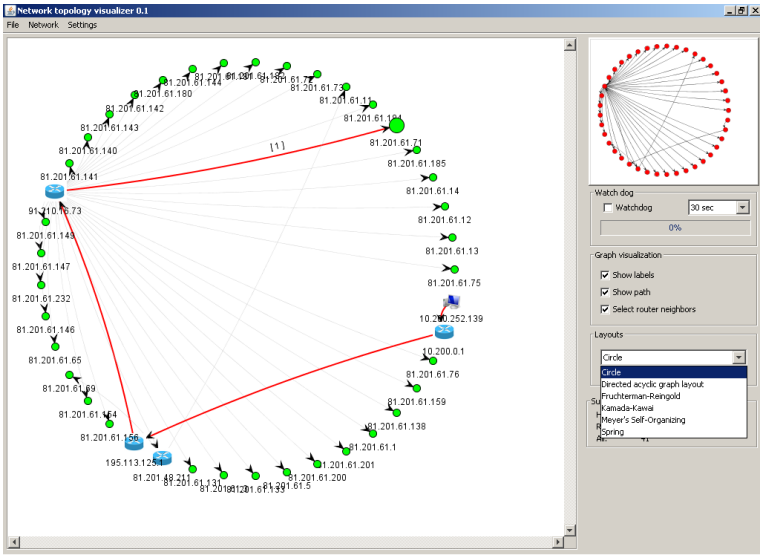
- **zoom in/out** – pomocí rolovacího kolečka myši,
- **rotace** – pomocí klávesy **shift** a stisku levého tlačítka myši,
- **roztažení** – pomocí klávesy **ctrl** a stisku levého tlačítka myši.



Obrázek 69 – výběr přímých sousedů

Dále je možné práci s modelem ovlivnit pomocí voleb z aplikační nabídky napravo.

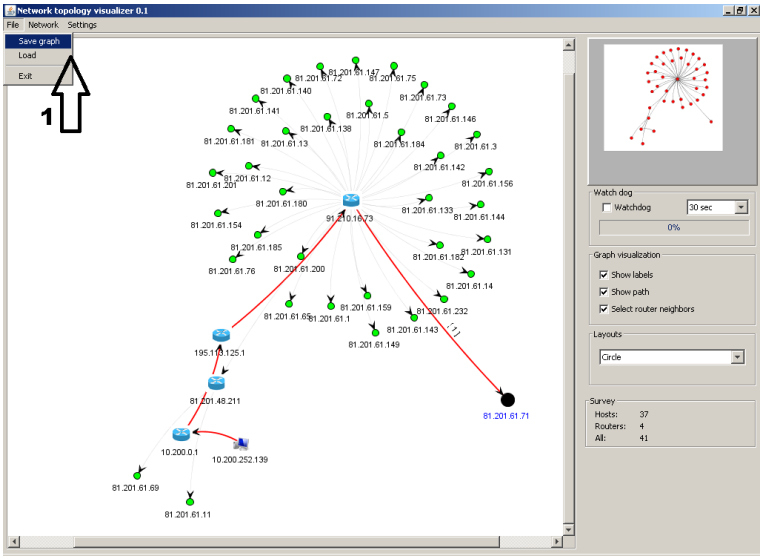
- **Show labels** – vypne/zapne zobrazování popisků.
- **Show path** – zobrazí trasu aktivní (uživatelem vybraný) uzel.
- **Select router neighbors** – po kliku levého tlačítka myši na směrovač se vyberou i jeho přímí následníci.
- **Watchdog** – zapne/vypne sledování dostupnosti.
- **Layouts** – změní režim rozložení.



Obrázek 70 – změna režimu rozložení na kruhový layout

Uložení do souboru

Model je možné nadále uložit případně načíst ze souboru XML.



Obrázek 71 – uložení modelu do souboru

Příloha D – obsah CD nosiče

Přiložené medium má dva hlavní adresáře:

- textova_cast – obsahuje textové dokumenty:
  - diplomova_prace.docx – text diplomové práce ve format word docx,
  - diplomova_prace.odt – text diplomové práce ve format open document,
  - diplomova_prace.pdf – text diplomové práce ve format pdf,
- prakticka_cast – obsahuje zdrojové soubory a knihovny,
  - bin32 – spustitelná verze aplikace (NetVisualizer) pro OS Windows 32bit,
  - bin64 – spustitelná verze aplikace (NetVisualizer) pro OS Windows 64bit,
  - NetVisualizer – zdrojové soubory diplomové práce a project v IDE NetBeans,
  - Libs – doplnkové knihovny,
    - WinPcap_4_1_3.exe – knihovna winpcap pro OS windows,
    - commons-net-3.2 – knihovny pro práci se sítí,
    - jpcap-0.7 – zdrojové kódy knihovny JPcap s aplikovanými záplatami,
    - jpcap_web – původní autorův web,
    - jung2 – zdrojové soubory knihovny JUNG2,
    - jung2-2_0_1 – binární podoba knihovny JUNG2,
    - jung2-2_0_1-apidocs.zip – Java dokumentace ke knihovně JUNG2,
  - depends22_x86.zip – nástroj Dependency Walker, odhalující chybějící knihovny.
  - depends22_x64.zip – nástroj Dependency Walker, odhalující chybějící knihovny.