Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Modely topologií počítačových sítí I

2015

Vlastimil Müller

Vedoucí práce: doc. Ing. Lenka Carr-Motyčková, CSc.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Vlastimil Müller
Název práce:	Modely topologií počítačových sítí I
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	doc. Ing. Lenka Carr-Motyčková, CSc.
Počet stran:	56
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Vlastimil Müller
Title:	Network topology models I
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Applied Computer Science, combined form
Supervisor:	doc. Ing. Lenka Carr-Motyčková, CSc.
Page count:	56
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Bakalářská práce se věnuje modelování topologií počítačových sítí. Práce se zabývá jak teorií abstraktních modelů topologií počítačových sítí, tak i jejich vlastní implementací v podobě aplikace `Nettop`*. Všechny abstraktní modely byly implementovány tak, aby jejich vlastnosti odpovídaly definicím z odborné literatury a textům věnovaným daným modelům. Uživatel aplikace* `Nettop` *je také schopen ovlivnit vlastnosti vygenerovaných topologií sadou vstupních parametrů, které odpovídají vlastnostem reálných počítačových sítí.*

Synopsis

This thesis is dedicated to network topology modeling. The aim of this this thesis is to study the theory of network topology models as well as their implementation in software application called `Nettop`*. Chosen abstract models were implemented so that their properties correspond with data from scientific literature and papers. User of the application is able to affect resulting topology properties through a set of input parameters which correspond with those of real life computer network.*

Klíčová slova: Internet; topologie; model; počítačová síť; graf

Keywords: Internet; topologie; model; computer network; graph

Rád bych tímto poděkoval doc. Ing. Lence Carr-Motyčkové, CSc. za ochotu spolupráce a odborné vedení této práce.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Seznam Pseudokódů

1 Úvod

Problematika modelování topologií počítačových sítí je velice rozsáhlá a sama o sobě má značný vliv na vývoj nových síťových protokolů, aplikací či hardwaru. Vědci se již delší dobu pokoušejí najít způsoby jak nejlépe modelovat topologie reálných sítí pouze za použití abstraktních modelů počítačových sítí. Důvody k modelování topologií počítačových sítí se odvíjí od vlastností počítačových sítí, které aktuálně potřebujeme studovat.

Tato práce studuje několik typů abstraktních modelů počítačových sítí a rozebírá jejich možnosti. Každý model je tak popsán i s jeho důležitými vlastnostmi tak, aby byly zřejmé přednosti toho daného modelu a možnosti jeho využití. Všechny pospané abstraktní modely byly zároveň implementovány v aplikaci `Nettop`, tak aby si uživatel mohl chování jednotlivých modelů a jejich vstupních parametrů vyzkoušet.

Práce je organizována do několika kapitol. Druhá kapitola se povrchně věnuje teorii počítačových sítí a zejména sítě Internet, kterou se abstraktní modely snaží často simulovat. Třetí kapitola je věnovaná právě teorii abstraktních modelů a tomu, jak pomocí grafů reprezentovat topologie počítačových sítí. Ve čtvrté kapitole je částečně popsána aplikace `Nettop` včetně částí zdrojového kódu, aby byl čtenář lépe schopen pochopit implementaci samotných modelů. Pátá kapitola se zabývá právě samotnou implementací vybraných abstraktních modelů a jejich vlastnostmi. Text bakalářské práce je zakončen závěrem s diskuzí o vlastnostech a využitelnosti jednotlivých modelů.

2 Počítačové sítě

Druhá kapitola se zabývá základním rozdělením počítačových sítí a zejména pak teorií sítě Internet, její strukturou a principy komunikace uvnitř sítě.

2.1 Co je to počítačová síť

Obecně se na počítačovou síť můžeme dívat jako na propojení několika počítačů za účelem vzájemné komunikace jejich uživatelů, sdílení dat, služeb a další. Počítačové sítě jako takové lze dělit podle několika kritérií, avšak nejčastější rozdělení je podle rozlohy dané sítě. [11] Z tohoto následně vyplývá realizace propojení v síti stejně jako typy a technologie použitých síťových prvků.

Lokální počítačová síť - LAN
V případě lokální počítačové sítě jde většinou o určité množství osobních počítačů propojených v rámci jedné místnosti, patra nebo celé firemní budovy. Spojení počítačů v sítích LAN bývá nejčastěji realizováno pomocí kroucené dvoulinky a v dnešní době již také velmi často bezdrátově.

Metropolitní počítačová síť - MAN
Rozloha těchto sítí často odpovídá velikosti celého města. Takovou sítí by mohla být třeba síť propojující všechny úřady městské samosprávy rozeseté po celém městě. V každé oblasti nebo budově pak realizace může vypadat jako klasická síť LAN, zatímco k propojení jednotlivých oblastí mezi sebou mohou sloužit vysokorychlostní optické kabely a veřejné komunikační sítě. citebi:prih

Rozlehlá počítačová síť - WAN
Rozlehlé sítě spojují místa od sebe často velmi vzdálená, ať už jsou to jednotlivá města či různé státy kdekoliv na Zemi. Počítače v takových sítích nebývají propojeny mezi sebou, ale za pomocí uzlových stanic, ke kterým se ostatní připojují. Informace tedy musí z výchozího počítače najít cestu přes jednu nebo více uzlových stanic ke svému cíli. Tyto cesty jsou zajišťovány síťovými protokoly a dalšími síťovými prostředky, tak aby cesta byla pokud možno co nejefektivnější. Spojení jednotlivých uzlových stanic bývá realizováno telefonními linkami, optickými kabely a za pomocí družicového spojení. Snad nejznámější světová WAN je síť Internet, některými detaily o její struktuře se bude text zabývat dále.

2.2 Síť Internet

2.2.1 Hierarchická struktura sítě Internet

Internet si z hlediska uspořádání jeho součástí můžeme hrubě představovat jako hierarchickou strukturu o dvou úrovních, přičemž horní úroveň by reprezentovala vzájemně propojené autonomní systémy a spodní úroveň by zobrazovala sítě uvnitř autonomních systémů jako takových.

2.2.2 Autonomní systémy - AS

Autonomní systémy neboli AS představují jednu nebo více sítí pod společnou technickou správou, která pak z hlediska sítě Internet vystupuje jako celek s jednotnou směrovací politikou. Toto výrazně zjednodušuje směrování v síti Internet vzhledem k tomu, že není potřeba, aby směrovací informace obsahovala údaje o každé síti a podsíti v Internetu, ale stačí pouze vědět, kde se nachází jednotlivé autonomní systémy a ty už si pak zajišťují komunikaci v rámci vlastní sítě samy. Pro identifikaci v síti Internet má každý autonomní systém přiděleno své vlastní 16-ti bitové, starší, nebo novější 32-bitové číslo. Identifikační čísla přiděluje organizace Internet Assigned Numbers Authority- IANA.

Ke komunikaci mezi sebou používají autonomní systémy jeden nebo více externích routerů komunikujících prostřednictvím Border Gateway Protokolu - BGP. Každý z těchto BGP routerů si udržuje směrovací tabulku autonomních systémů, do kterých je přes tento router možno přistupovat. Směrovací tabulka také obsahuje seznam cest k ostatním autonomním systémům. Cesta k autonomnímu systému zjednodušeně vypadá jako sled identifikačních čísel těch systémů, přes které je potřeba dojít až k cílovému autonomnímu systému.

2.2.3 Border Gateway Protokol - BGP

Border Gateway Protokol neboli BGP zajišťuje směrování mezi autonomními systémy v síti Internet nebo v jiných velkých sítích, které mohou být rozděleny na více segmentů. BGP poskytuje mechanismy k zajištění distribuce cest mezi propojenými autonomními systémy, zatímco volbu cesty ponechává na správci dané autonomní sítě. Výběr cesty tak záleží na politice směrování každého autonomního systému a je velmi často ovlivňován různými kritérii mezi něž patří například délka a cena cesty, upřednostňované autonomní systémy přes které lze data přenášet stejně jako ty systémy, kterým se vyhnout.

Úplnou směrovací informaci si mezi sebou autonomní systémy vymění hned při prvním navázání spojení a poté si už vyměňují jen cesty, které se ve směrovacích tabulkách změnily. Ve směrovacích tabulkách se ukládají vždy pouze ty nejvhodnější cesty a ty jsou pak dále propagovány sousedním AS, které je opět podle vlastní politiky zařadí do svých tabulek nebo je odstraní. Díky tomu BGP také pomáhá zabraňovat vzniku nechtěných směrovacích smyček.

Na základě informací z BGP lze dále vytvářet grafické znázornění propojitelnosti autonomních systémů a vytvářet tak představu o topologii sítě Internet.

2.2.4 Topologie sítě Internet

Na topologii sítě Internet se dá pohlížet z různých úhlů pohledu a studovat tak odlišné vlastnosti. U sítě Internet můžeme například studovat jak vlastnosti chování síťových protokolů mezi autonomními systémy, tak jejich chování uvnitř samotných autonomních systémů. Obě jsou neméně důležité avšak většina studií se zabývá spíše chováním topologií při komunikaci mezi samotnými autonomními systémy než mezi routery uvnitř AS a to z několika důvodů:

- na úrovni AS je k dispozici nejvyšší úroveň abstrakce
- je relativně snadné k ní získat potřebná data z BGP tabulek
- ostatní úrovně topologie jsou na AS částečně závislé

Je potřeba podotknout, že topologie sítě Internet na úrovni AS založená na sběru dat z BGP routerů jednotlivých AS nemusí reflektovat reálnou topologii. Toto je zapříčiněno tím, že každý AS si sám určuje svoji směrovací politiku, která může být ovlivněna různými faktory, a proto se nemusí v sebraných datech objevit některé spoje, které sice fyzicky mohou existovat, ale daný AS je nevyužívá.

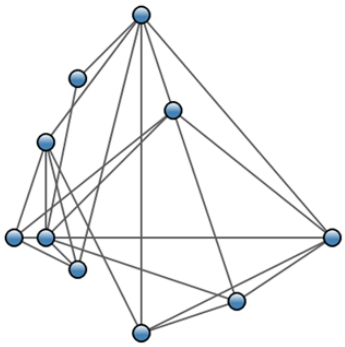
3 Modely topologií počítačových sítí

Při vývoji a výzkumu nových síťových prvků, protokolů nebo aplikací je potřeba je řádně otestovat ještě před jejich zavedením, aby se tak předešlo případným problémům. Abstraktní modely topologií počítačových sítí nám tedy mohou sloužit právě k testovacím účelům nebo například ke sledování vývoje a růstu různých sítí. Toto bychom na vlastních počítačových sítích nebyli schopni pořádně realizovat. Od modelů se očekává především co největší schopnost přesně zachytit vlastnosti a chování reálných sítí.

Za dobu vývoje bylo vyvinuto množství modelů, které se od sebe liší jak vlastnostmi, které se snaží zachytit, tak i složitostí a náročností na hardware. Přibližně by se daly rozdělit do několika kategorií, kterým se věnuje odborná literatura [2] [3].

3.1 Náhodné modely

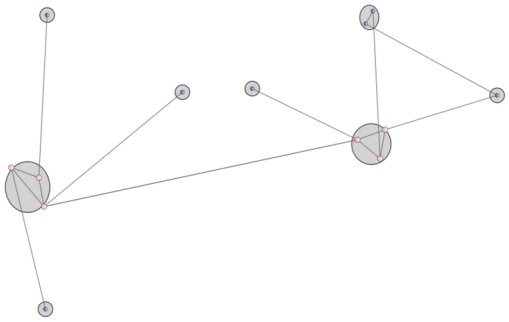
Náhodné modely rozmisťují uzly náhodně a spojování uzlů hranami je řízeno pravděpodobnostními funkcemi. Tyto funkce mohou být čistě náhodné nebo mohou záviset na různých parametrech jako například vzdálenost uzlů mezi sebou. Funkce postupně vypočítá pro každou dvojici uzlů pravděpodobnost, s jakou budou spojeny hranou a jestliže je tato pravděpodobnost vyšší než porovnávaná hodnota, jsou oba uzly spojeny hranou. Porovnávaná hodnota se u různých implementací a variant jednotlivých modelů může lišit. Mezi tyto modely řadíme například Erdös-Rényi a Waxman modely.



Obrázek 1: Graf vygenerovaný náhodným algoritmem Waxman.

3.2 Hierarchické modely

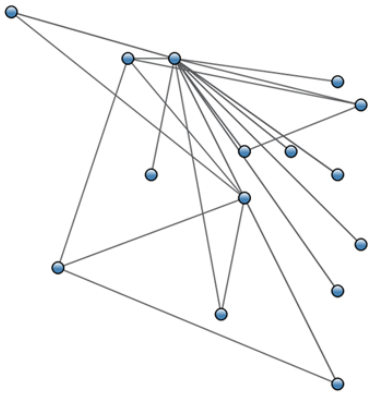
Hierarchické modely, mimo jiné vlastnosti, zachycují hlavně strukturu sítě a její rozdělení do úrovní. Úrovně pohledu na topologii sítě se věnovala podkapitola 2.2.4 Topologie sítě Internet. Jsou vhodné zejména ke sledování vlastností protokolů a směrování mezi doménami. Pomocí hierarchických modelů je možné např. testovat reálná data z BGP. Hierarchické modely mohou pro generování jednotlivých svých úrovní využívat i jiné, často právě náhodné, modely. Mezi zástupce těchto modelů patří Tiers a Transit-Stub.



Obrázek 2: Graf vygenerovaný hierarchickým algoritmem Transit-Stub.

3.3 Dynamický růst

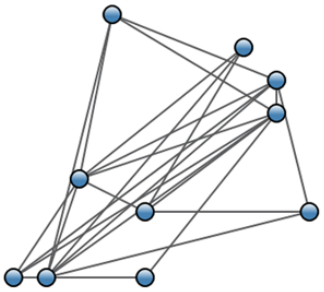
Zde je nejdříve vygenerován menší graf, ke kterému jsou později připojovány další uzly. Tyto modely se snaží zachytit klíčovou vlastnost, kdy nově připojované uzly mají tendenci se připojit k uzlům s vyšším stupněm uzlu. Takto vznikne v grafu několik uzlů s vysokým stupněm a zbytek uzlů s jen velmi málo spojeními. K zástupcům těchto modelů patří Barabási-Albert a GLP modely.



Obrázek 3: Graf vygenerovaný algoritmem Barabási-Abert.

3.4 Mocninné zákony

Mocninné zákony u velkých počítačových sítí a zejména sítě Internet poprvé pozorovali a následně popsali bratři Faloutsové. Ve svých článcích [1] [2] ukázali, že právě topologie sítě Internet a vztahy mezi autonomními systémy může být popsány třemi mocninnými zákony.



Obrázek 4: Graf vygenerovaný algoritmem PLOD dodržujícího mocninné zákony.

3.4.1 Zákon řádu

Stupeň d_v, uzlu v je úměrný řádu tohoto uzlu, r_v umocněného konstantou R:

$$d_v \propto r_v^R \tag{1}$$

Řádem uzlu r_v se rozumí pořadí uzlu v mezi uzly grafu seřazenými sestupně podle jejich stupně d_v.
Při testech zákon řádu vykazoval pro některé skupiny hodnot r_v a d_v stejné nebo velmi blízké hodnoty R. Zákon tedy může sloužit jako nástroj k dělení grafů do skupin podle typu. Zároveň zákon řádu ukázal, že odráží i způsob jakým se domény mezi sebou mohou spojovat.

3.4.2 Zákon stupně

Frekvence f_d stupně d, je úměrná stupni d umocněnému konstantou σ:

$$f_d \propto d^\sigma \tag{2}$$

Tento zákon nahlíží na rozdělení stupňů v sítí Internet jako na nenáhodné. Bylo zpozorováno, že nižší stupně jsou častější a tuto vlastnost zachycuje exponent σ. Pokud se tedy exponent σ vygenerované topologie značně liší od hodnot reálných sítí, pravděpodobně nejde o realistickou topologii.

3.4.3 Zákon vlastního čísla

Vlastní čísla λ jsou úměrné pořadí i, umocněného konstantou ε:

$$\lambda \propto i^{\varepsilon} \tag{3}$$

Pořadí i značí pořadí daného vlastního čísla $lambda_i$ mezi ostatními vlastními čísly, která jsou seřazena sestupně podle jejich velikosti.
Vlastními čísly grafu se rozumí vlastní čísla jeho matice sousednosti. Množství literatury dokazuje, že vlastní čísla grafu jsou úzce spjatá se základními vlastnostmi, jako je průměr grafu, počet hran, počet cest a další.

3.5 Reprezentace modelu

K tomu, abychom mohli analyzovat chování a vývoj počítačových sítí, a to zejména těch rozsáhlých, potřebujeme mít možnost sítě ve zjednodušené formě nějak reprezentovat. Sítě jsou pro tyto účely reprezentovány grafy. Některé aspekty reprezentace počítačových sítí grafy si rozebereme dále v textu.

3.5.1 Graf jako počítačová síť

Využití grafu pro reprezentaci počítačové sítě je v celku zřejmé. Uzly grafu, v závislosti na typu sítě a zkoumaných vlastnostech, představují buďto samotné počítače, routery nebo třeba i celé autonomní systémy. Hrany grafu tedy zcela analogicky reprezentují spojení mezi těmito prvky. Většina grafů, kromě specifických případů, bývá neorientovaná. To je logické i vzhledem k tomu, že i komunikace v síti je z pravidla možná oběma směry. Často však bývají tyto grafy vážené, kdy se jako váhy jednotlivým hranám mohou přisuzovat délky skutečných spojů, rychlost nebo kapacita linky a tak podobně.

Grafické stránce reprezentace sítě nejsou kladeny žádné meze, avšak vzhledem k názornosti bývají tyto aplikace v celku jednoduché a hlavně účelné. Využívají se zejména základní geometrické tvary pro reprezentaci uzlů a plné úsečky pro hrany grafu. To vše bývá doplněno relevantními popisky a daty, případně také nějakými statistickými nástroji pro vyhodnocování údajů o grafu.

3.5.2 Vlastnosti grafů

Tato část textu popisuje zejména ty vlastnosti grafů [4] [9], které souvisí s modelováním topologií počítačových sítí. V této části textu je graf značen $G(V, E)$, kde V je množina uzlů a E je množina hran v grafu.

Stupeň uzlu
Stupněm uzlu u grafu $G=(V,E)$ se rozumí počet hran, které s uzlem u incidují.

Sousední uzly
Uzly označíme jako sousední pokud jsou spojeny hranou.

Souvislost
Graf $G = (V, E)$ je souvislý, jestliže pro každé jeho dva uzly u a v existuje v G cesta z u do v.

Úplnost
Neorientovaný graf $G = (V, E)$ je úplný pokud jsou každé jeho dva uzly spojené hranou. Úplný graf je maximálně souvislý. Úplné grafy značíme K_n.

Průměr grafu
Průměrem grafu rozumíme nejkratší cestu mezi dvěma nejvzdálenějšími uzly v grafu. V praxi to znamená najít všechny nejkratší cesty v grafu a z nich vybrat tu nejdelší. Pokud je graf nesouvislý, je jeho průměr nekonečno.

Hustota grafu
O grafu *G=(V,E)* řekneme, že je hustý, jestliže se počet hran v grafu blíží jejich maximálnímu možnému počtu. Maximální hustota grafu je 1, minimální je 0. Hustotu neorientovaného grafu vypočteme:

$$D = \frac{2|E|}{|V|(|V| - 1)} \tag{4}$$

kde:

$|E|$ = počet hran v grafu
$|V|$ = počet uzlů v grafu

Průměrný stupeň uzlu
Průměrný stupeň uzlu udává kolik je v grafu *G=(V,E)* hran v porovnání k počtu uzlů:

$$k = \frac{1}{|V|} \sum_{i=1}^{|} V|k_i = \frac{2|E|}{|V|} \tag{5}$$

kde:

$|E|$ = počet hran v grafu
$|V|$ = počet uzlů v grafu

4 Aplikace Nettop

Součástí zadání bakalářské práce je naprogramování generátoru grafů znázorňujících topologie počítačových sítí. Jako způsob prezentace modelů, které jsem vybral k implementaci jsem zvolil aplikaci s grafickým uživatelským rozhraním. Grafické uživatelské rozhraní umožňuje názorné a přehledné zobrazení vygenerovaných grafů. Grafické uživatelské rozhraní také poskytuje větší možnosti zobrazení statistických dat než například aplikace využívající výhradně příkazový řádek. Aplikace je pojmenována Nettop jako spojení dvou anglických slov a to `Network` a topology.

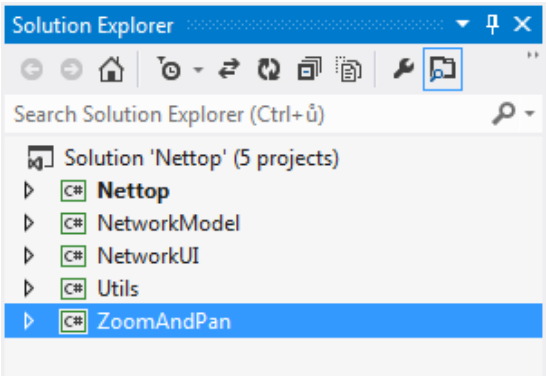
Účelem aplikace `Nettop` je generování a následná vizualizace grafů sítí s využitím abstraktních modelů. Uživatel si podle zvoleného typu modelu může nastavit klíčové parametry, které ovlivňují výsledný graf sítě. Vstupní parametry grafů byly zvoleny podle dostupné dokumentace k jednotlivým grafům a také podle jejich následné implementace. I přes to, že se odborné texty často nevěnují vlastní realizaci modelů, byly implementovány tak, aby výsledné grafy svými vlastnostmi co nejvíce odpovídaly charakteristikám popsaným v odborných dokumentech.

4.1 Struktura aplikace

`Nettop` je napsán v jazyce C# v prostředí MS Visual Studio 2012 využitím technologie Windows Presentation Foundation (WPF). Zdrojový kód byl napsán podle návrhového modelu MVVM. Detaily tohoto návrhového modelu diskutuje [5].

Aplikace je částečně založena na kódu [6], ze kterého přebírá zejména část uživatelského rozhraní, která má na starosti prostředí pro zobrazování grafu a jeho funkce přiblížení a navigace. Užití tohoto kódu je v souladu s licencí, která je rovněž uvedena na [6]. Zbytek kódu aplikace, včetně implementace všech modelů topologií a podpůrných metod pro tyto modely, byl vytvořen a upraven mnou podle potřeb zadání práce. Každý zdrojový soubor aplikace obsahuje hlavičku se zmínkou o autorství kódu v daném souboru a případných úpravách kódu původního.

Samotné řešení aplikace sestává z pěti samostatných projektů jak ukazuje obrázek 5



Obrázek 5: Projekty obsažené v řešení aplikace

Nettop je samotná aplikace. Obsahuje třídy pro všechny modely topologií, uživatelské rozhraní a další nástroje aplikace.

NetworkView obsahuje ViewModel třídy pro projekt `Nettop`. Zejména třídy uchovávající stavy uzlu, hrany a sítě.

NetworkUI obsahuje vlastní ovládací prvek pro práci s grafem. Také definuje, které prvky uživatelského rozhraní budou reprezentovat datové struktury pro uzly a hrany.

Utils a ZoomAndPan obsahují pomocné metody pro `NetworkUI`.

4.2 Důležité třídy a metody

Níže jsou popsány vybrané třídy ze zdrojového kódu aplikace `Nettop`. Třídy byly vybrány tak, aby v následující kapitole neměl čtenář problémy s porozuměním ukázek implementace abstraktních modelů. Třídy nebo metody specifické pro určité modely budou rozebrány v kapitolách věnujícím se právě těmto modelům.

4.2.1 Třída Nettop.MainWindow

V třídě `MainWindow` je definováno celé uživatelské rozhraní aplikace. Konkrétně obsahuje metody pro řízení generování grafů a některé vstupně-výstupní operace jako ukládání, načítání a export grafu do souboru.

Protože generování grafu probíhá ve vlastním vlákně, je v konstruktoru třídy uveden odkaz na ViewModel, který udržuje aktuální data grafu. Následně jsou na události vlákna pro generování grafů navázány metody pro jejich obsluhu tak jak ukazuje zdrojový kód 1

```
1 public MainWindow()
2 {
3         ...
4         //odkaz na ViewModel s aktuálními daty o grafu
5         this.ViewModel.NetworkView = networkControl;
6
7         //generování grafu běží v samostatném vlákně
8         modelThread = new BackgroundWorker { WorkerReportsProgress = true,
              WorkerSupportsCancellation = true };
9         modelThread.DoWork += new DoWorkEventHandler(modelThread_DoWork);
10        modelThread.RunWorkerCompleted += new
              RunWorkerCompletedEventHandler(modelThread_RunWorkerCompleted);
11        modelThread.ProgressChanged += new
              ProgressChangedEventHandler(modelThread_ProgressChanged);
12        ...
13 }
```

Zdrojový kód 1: Konstruktor třídy Nettop.MainWindow

Metoda modelThread_DoWork

Z metod obsluhujících generování modelů je metoda `modelThread_DoWork` nejpodstatnější, protože volá metody generující jednotlivé grafy. Modely získávají vstupní parametry z ovládacího prvku v uživatelském rozhraní. Pro každý model je implementován zvláštní ovládací prvek s posuvníky pro nastavení vstupních parametrů modelu. Tyto ovládací prvky jsou popsány v podkapitole 4.3.1 Uživatelské rozhraní a funkce aplikace Nettop

```
1 private void modelThread_DoWork(object sender, DoWorkEventArgs e)
2 {
3         //vytvoření nové instance BackgroundWorkeru
4         BackgroundWorker worker = sender as BackgroundWorker;
5         switch ((int)e.Argument)
6         {
7                 ...
8                case 2:
9                     //vytvoření nové instance daného Modelu
10                    modelWaxman = new ModelWaxman();
11                    //u všech modelů se generování grafu spouští metodou nazvanou
                          RunModel, parametry jsou předávány z UI
12                    e.Result = modelWaxman.RunModel(worker,
                          this.waxmanSetup.Nodes, this.waxmanSetup.XMin,
                          this.waxmanSetup.XMax, this.waxmanSetup.YMin,
                          this.waxmanSetup.YMax, this.waxmanSetup.Alfa,
                          this.waxmanSetup.Beta, this.waxmanSetup.NodeSpan);
13                    break;
14                case 4:
15                ...
16                //varianty pro další abstraktní modely
17                ...
18        }
19 }
```

Zdrojový kód 2: Metoda řídící generování grafů

4.2.2 Třída Nettop.MainWindowViewModel

Tato třída uchovává aktuální data o grafu, který byl vygenerován a umožňuje přístup k jednotlivým parametrům a vlastnostem tohoto grafu. Jak ukazuje zdrojový kód 3, obsahuje také několik proměnných ovlivňujících úroveň úpravy grafů.

```
1 public MainWindowViewModel()
2 {
3         //proměnná obsahující samotný graf
4         network = null;
5         //proměnné ovlivňující ''plochu'' grafu a její změnu při přblížení nebo
              oddálení
6         contentScale = 1;
7         contentOffsetX = 0;
8         contentOffsetY = 0;
9         contentWidth = 1000;
10        contentHeight = 1000;
11        contentViewportWidth = 0;
12        contentViewportHeight = 0;
13        //pravdivostní proměnná, která zapíná/vypíná ovládací prvky uživatelského
              rozhraní
14        isUIEnabled = false;
15        //povoluje přidávání uzlů
16        canAddNodes = false;
17        //povoluje
18        canEdit = false;
19        //povoluje odebírání uzlů
20        canRemoveNodes = false;
21        //povoluje posouvání uzlů myší
22        canMoveNodes = false;
23 }
```

Zdrojový kód 3: Konstruktor třídy Nettop.MainWindowViewModel

Vlastnost Network

Vlastnost `Network` třídy `MainWindowViewModel` je důležitá zejména z toho důvodu, že, jak její název napovídá, uchovává v sobě údaje grafu. Vlastní datové struktuře, která tyto informace uchovává, se budu věnovat dále v textu. Zdrojový kód 4 ukazuje přístupové metody této vlastnosti.

```
1 public MainWindowViewModel()
2 {
3         public NetworkViewModel Network
4         {
5             get { return network; }
6             set
7             {
8                 //přiřazení grafu do proměnné
9                 network = value;
10
11                //aktivovat ovládací prvky UI, jakmile je graf přiřazen
12                this.isUIEnabled = true;
13                //Spustí metodu, která nastaví velikost pozadí, na kterém je graf
                      zobrazen
14                SetContentSize();
15
16                //každému uzlu v grafu přiřadíme obsluhu události
                      PropertyChanged, tak abychom mohli kontrolovat a měnit jejich
                      stavy
17                foreach(NodeViewModel n in this.network.Nodes)
18                {
19                    n.PropertyChanged += new
                          PropertyChangedEventHandler(node_PropertyChanged);
20                }
21                //oznámení změny této vlastnosti
22                OnPropertyChanged("Network");
23            }
24        }
25 }
```

Zdrojový kód 4: Vlastnost Network třídy Nettop.MainWindowViewModel

4.2.3 Třída Nettop.OverviewWindow

Jedná se o zjednodušenou třídu `Nettop.MainWindow`, bez jakékoliv kontroly nad samotným grafem. Slouží jako náhledové okno pro navigaci zejména při větších rozměrech grafu nebo při přiblížení.

4.2.4 Třída Nettop.NetworkStatsWindow

Tato třída slouží pro zobrazení okna se statistickými údaji o právě vygenerovaném grafu. Údaje zobrazené v okně se různí v závislosti na typu grafu. Tyto údaje jsou uloženy v samotné vlastnosti `Nettop.Network`.

4.2.5 Třída NetworkModel.NodeViewModel

V této třídě jsou uchovávány vlastnosti jednotlivých uzlů. Implementace uzlů obsahuje také připojovací body, ke kterým jsou hrany vlastně připojeny. Aplikace dovoluje mít vstupní a výstupní připojovací body pro jeden uzel avšak pro zjednodušení zobrazení byly sloučeny do jednoho. Nejvýznamnější vlastnosti i s jejich popisem ukazuje zdrojový kód 5

```
1               ...
2       //pravdivostní hodnota udává, zda je uzel vstupním uzlem do tranzitní
            domény
3       public bool IsTransitGate { ... }
4
5       //pravdivostní hodnota udává, zda je uzel členem tranzitní domény
6       public bool IsTransit { ... }
7
8       //pravdivostní hodnota udává, zda je uzel členem koncové domény
9       public bool IsStub { ... }
10
11      //pravdivostní hodnota udává, zda je uzel sousedem jiného
12      //využívá se například při označování hran v UI
13      public bool IsAdjacent { ... }
14
15      //vrací nebo zapisuje jméno uzlu
16      public string Name { ... }
17
18      //pravdivostní hodnota udává zda je uzel označen
19      public bool IsSelected { ... }
20
21      //vrací nebo zapisuje souřadnici uzlu na ose X
22      public double X { ... }
23
24      //vrací nebo zapisuje souřadnici uzlu na ose Y
25      public double Y { ... }
26      ...
27      //udává velikost uzlu
28      //tato hodnota je dána velikostí prvku, který reprezentuje uzel v
            uživatelském rozhraní
29      public Size Size { ... }
30      ...
31      //vrací seznam hran, které jsou spojeny s uzlem
32      public ICollection<ConnectionViewModel> AttachedConnections { ... }
33              ...
```

Zdrojový kód 5: Vybrané vlastnosti třídy NetworkModel.NodeViewModel

4.2.6 Třída NetworkModel.ConnectionViewModel

Zde jsou uchovávány veškeré důležité vlastnosti pro hrany grafu. Z hlediska implementace není hrana v grafu přímo mezi dvěma uzly, ale mezi jejich připojovacími body. Zdrojový kód 6 opět vybírá ty nejdůležitější vlastnosti.

```
1               ...
2       //možné grafické reprezentace hran
3       //čára zakončená šipkou na jednom konci, čára se šipkami na obou koncích,
            čára bez šipek
4       public enum ConnectionType {
5           Simple,
6           Double,
7           Unoriented
8       }
9       //pravdivostní hodnota udává zda byl označen některý z připojených uzlů
10      //slouží pro označování v UI
11      public bool IsRootNodeSelected { ... }
12              ...
13      //počáteční připojovací bod
14      public ConnectorViewModel SourceConnector { ... }
15      //cílový připojovací bod
16      public ConnectorViewModel DestConnector { ... }
17              ...
```

Zdrojový kód 6: Vybrané vlastnosti třídy NetworkModel.ConnectionViewModel

4.2.7 Třída NetworkModel.GroupViewModel

Tato třída slouží zejména pro potřeby uživatelského rozhraní. V případě, že je potřeba označit skupinu uzlů, která patří například do stejné domény, je možno k tomu využít právě třídu `GroupViewModel`. Uživatelské rozhraní pak na pozici instance této třídy vykreslí předdefinovaný obrazec reprezentující danou skupinu uzlů.

4.2.8 Třída NetworkModel.NetworkViewModel

Třída `NetworkViewModel` slučuje dohromady třídy reprezentující uzly, hrany a skupiny tak, aby společně reprezentovali jeden graf. Tato třída má celkem pochopitelně nejvíc vlastností a to proto, že uchovává nejenom kolekce uzlů a hran, ale i statistické údaje, které jsou zobrazovány v okně statistik.

```
1           ...
2       //vrací průměrný stupeň uzlu v grafu
3       public double AvgNodeDeg { ... }
4
5       //vrací maximální stupeň uzlu v grafu
6       public double MaxNodeDeg { ... }
7
8       //vrací minimální stupeň uzlu v grafu
9       public double MinNodeDeg { ... }
10
11      //vrací průměr grafu
12      public double Diameter { ... }
13
14      //vrací počet tranzitních domén
15      //pouze u modelu Transit-Stub
16      public double TransitDomainsNumber { ... }
17
18      //vrací počet koncových domén
19      //pouze u modelu Transit-Stub
20      public double StubDomainsNumber { ... }
21
22      //vrací počet hran mezi tranzitními a koncovými doménami
23      //pouze u modelu Transit-Stub
24      public double ConnectionsTS {... }
25
26      //vrací počet hran mezi koncovými doménami
27      //pouze u modelu Transit-Stub
28      public double ConnectionsSS { ... }
29
30      //vrací hodnotu hustoty grafu
31      public double Density { ... }
32
33          ...
34
35      // vrací nebo zapisuje šířku grafu (rozdíl mezi nejnižší a nejvyšší
            hodnotou na ose X)
36      public double Width { ... }
37
38      // vrací nebo zapisuje výšku grafu (rozdíl mezi nejnižší a nejvyšší
            hodnotou na ose Y)
39      public double Height { ... }
40
41          ...
42
43      //kolekce obsahující všechny uzly grafu
44      public ImpObservableCollection<NodeViewModel> Nodes { ... }
```

```
45
46          //kolekce obsahující všechny skupiny uzlů
47          public ImpObservableCollection<GroupViewModel> NodeGroups { ... }
48
49                  ...
50
51          //kolekce obsahující všechny hrany uzlu
52          public ImpObservableCollection<ConnectionViewModel> Connections { ... }
53
54          //udává pomyslný střed grafu
55          public Point NetworkCentre { ... }
56                  ...
```

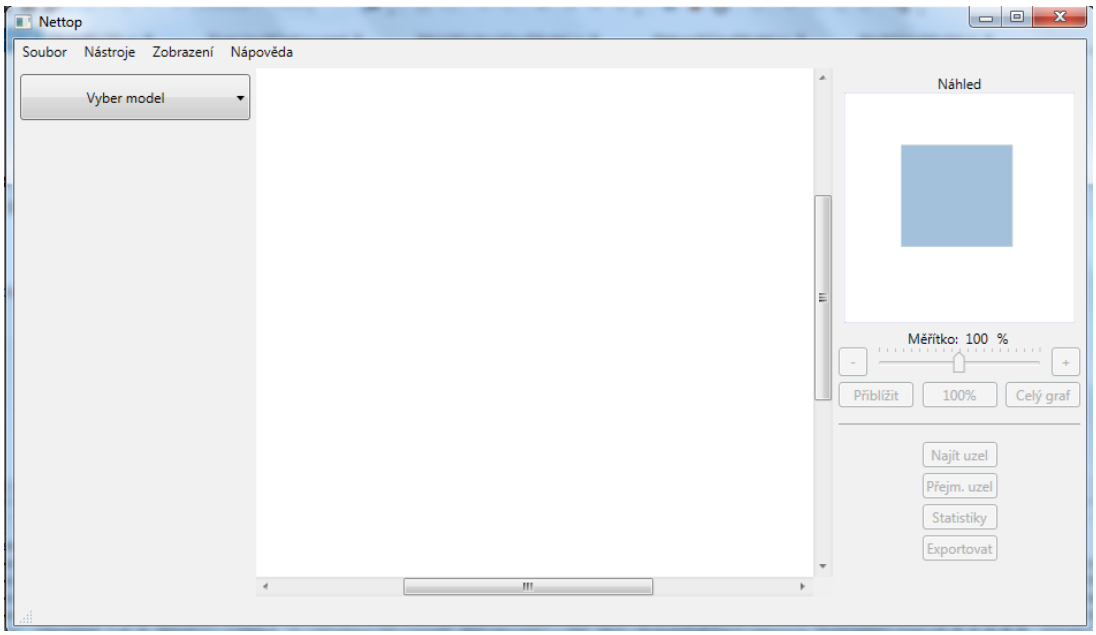
Zdrojový kód 7: Vybrané vlastnosti třídy NetworkModel.NetworkViewModel

4.3 Uživatelské rozhraní a jeho funkce

Použití technologie WPF umožňuje vytvářet bohaté uživatelské rozhraní díky tomu, že k jeho popisu využívá jazyk XAML. XAML dovoluje přehlednou a relativně snadnou organizaci kódu uživatelského rozhraní, který je oddělen od samotného kódu aplikace. To usnadňuje vývoj a následné testování aplikace a odstraňování chyb.

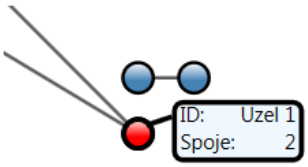
4.3.1 Uživatelské rozhraní a funkce aplikace Nettop

Uživatelské rozhraní aplikace Nettop jsem navrhl tak, aby měl uživatel většinu ovládacích prvků ihned po ruce. Jedno primární okno obsahuje pracovní plochu pro zobrazování vygenerovaných grafů, ovládací prvky pro nastavování vstupních parametrů a další nástroje pro práci s vygenerovanými grafy.

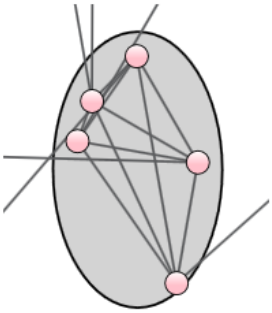


Obrázek 6: Primární okno aplikace Nettop

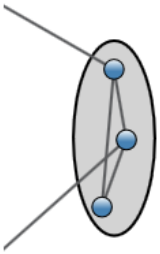
Grafy v uživatelském rozhraní
Třetí kapitola této práce uvádí, že počítačové sítě reprezentujeme pomocí grafů. V grafickém uživatelském rozhraní mohou mít grafy různou podobu, avšak měly by zůstat co nejpřehlednější. V aplikaci Nettop jsou proto uzly reprezentovány kruhy a hrany jednoduchými čarami. Uzly reagují na kliknutí myši změnou výplně kruhu na červenou barvu a zobrazením názvu uzlu a počtu hran, které s daným uzlem incidují viz obrázek 7



Obrázek 7: Reprezentace uzlů a hran



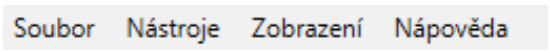
Obrázek 8: Reprezentace tranzitní domény



Obrázek 9: Reprezentace koncové doména

4.3.2 Ovládací prvky uživatelského rozhraní

Nabídkový pruh
Hlavní nabídkový pruh ovládá především dodatečné funkce aplikace Nettop a je rozdělen do čtyř kategorií:



Obrázek 10: Hlavní nabídkový pruh aplikace Nettop

Soubor Obsahuje položky, které ukládají a načítají vygenerované grafy do a ze souboru. Přes toto menu je také možné ukončit aplikaci Nettop.

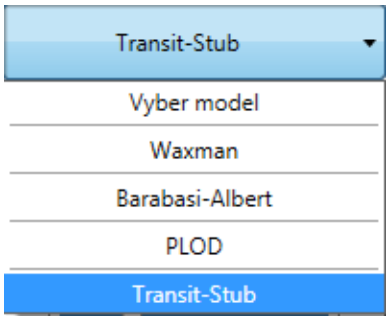
Nástroje Nástroje pro vyhledávání a přejmenování uzlů v grafu. Také je zde položka, která zobrazí sekundární okno se statistickými údaji o grafu.

Zobrazení Obsahuje položky, které zapínají nebo vypínají zobrazování různých informativních popisků v grafu.

Nápověda Obsahuje položku Nápověda, která zobrazí dialogové okno se souborem nápovědy.

Výběr modelu topologie

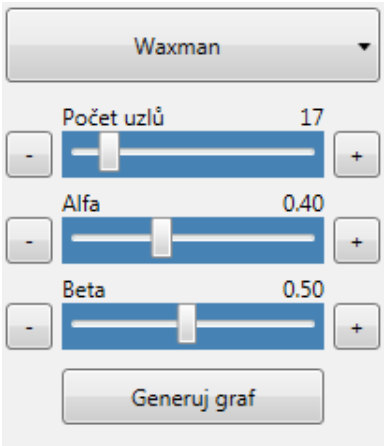
K výberu modelu topologie, který chce uživatel vygenerovat, slouží rozbalovací seznam umístěný na levém okraji primárního okna. Poté, co si uživatel vybere model podle něhož chce vygenerovat graf sítě, se pod tímto seznamem zobrazí ovládací prvek s nastavením vstupních parametrů zvoleného modelu.



Obrázek 11: Výběr modelu topologie

Vstupní parametry modelů

Každý model implementovaný v aplikaci Nettop má vlastní ovládací prvek, který slouží pro nastavení jeho vstupních parametrů. Na obrázku 12 je vidět, že parametry nastavujeme pomocí posuvníku a tlačítek + a −. Rozsahy nastavitelných hodnot na posuvníku byly zvoleny s ohledem na data uvedená v odborných textech popisujících jednotlivé modely. Ohled jsem také bral na dosažení přiměřené doby generování grafu.



Obrázek 12: Ovládací prvek se vstupními parametry modelu Waxman

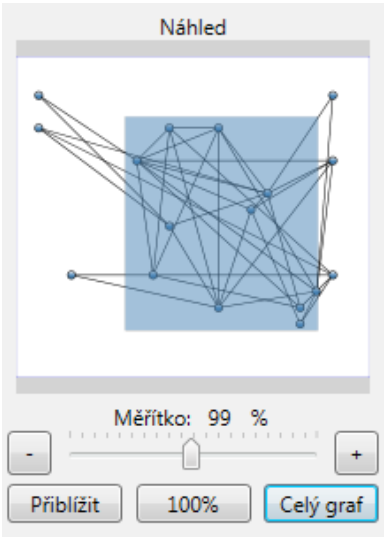
Práce s pracovní plochou
Pracovní plocha slouží ke zobrazování a navigaci v grafu. Uživatel si může také graf přiblížit nebo oddálit což je užitečné při větší hustotě grafu nebo pokud graf zabírá větší plochu.
Navigovat v pracovní ploše grafu se může uživatel podržením pravého tlačítka myši a pohybem do stran nebo pomocí posuvníků na okraji pracovní plochy.

Pro orientaci v rozsáhlejších grafech slouží navigační okno u pravého okraje primárního okna. Pod navigačním oknem jsou umístěny ovládací prvky pro nastavení měřítka zobrazeného grafu a úrovně přiblížení. Pro práci s měřítkem zobrazeného grafu slouží také tři tlačítka:

Přiblížit Přiblíží vybraný uzel. Pokud není vybraný žádný uzel zobrazí celý graf.

100% Nastaví měřítko na původních 100%

Celý graf Nastaví měřítko tak, že bude zobrazen celý graf.



Obrázek 13: Navigační okno s ovládacími prvky měřítka

4.4 Funkce aplikace Nettop

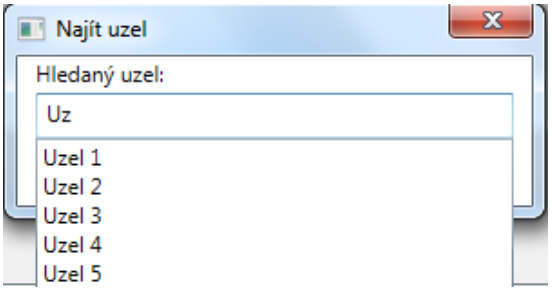
Zde popisuji některé z funkcí aplikace Nettop usnadňující práci s vygenerovanými grafy. Tyto funkce jsou dostupné buďto z hlavního nabídkového pruhu anebo pomocí tlačítek umístěných pod navigačním oknem aplikace viz obrázek 6

Načtení a uložení grafu ze souboru
Aplikace Nettop umožňuje vygenerované grafy ukládat a následně načítat ze souboru ve formátu .xml. Grafy uložené v souboru .xml jsou přehledné a některé základní údaje o grafu může uživatel vyčíst přímo z nich. Ačkoli je to možné nedoporučuji, aby se uživatel pokoušel uložené grafy ve větším rozsahu upravovat, protože by při pokusu o načtení do aplikace nemuseli projít kontrolou správnosti.

Export grafu do obrazového souboru
Tato funkce exportuje vygenerovaný graf do obrazového souboru ve formátu .png. Rozlišení souboru .png je dáno velikostí grafu a určuje jej aplikace sama.

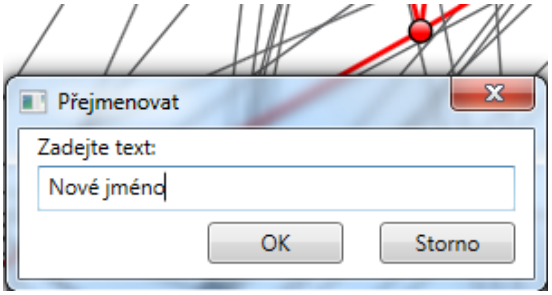
Ukončení aplikace
Aplikaci je možné ukončit buďto z hlavního nabídkového pruhu nebo pomocí tlačítka v pravém horním rohu rámu primárního okna. Potvrzení ukončení vyžaduje aplikace přes dialogové okno.

Vyhledání uzlu v grafu
Zobrazí dialogové okno s textovým polem, do kterého uživatel zadá název vyhledávaného uzlu. Aplikace se poté pokusí hledaný uzel v grafu najít a přiblížit jej. Pro usnadnění práce s vyhledáváním nabízí textové pole nápovědu s názvy všech uzlů v grafu.



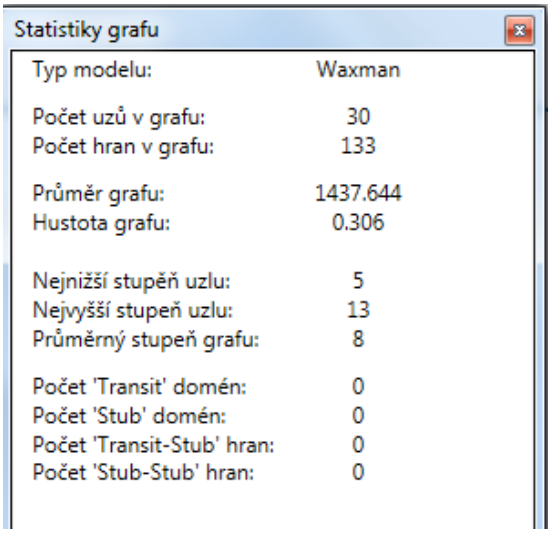
Obrázek 14: Dialogové okno pro vyhledání uzlu v grafu

Přejmenování uzlu
Při označení libovolného uzlu zobrazí dialogové okno s textovým polem, do kterého může uživatel napsat požadovaný nový název uzlu.



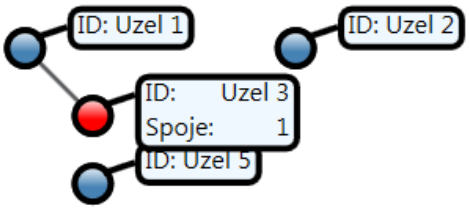
Obrázek 15: Dialogové okno pro přejmenování uzlu

Statistické údaje
Zobrazí sekundární okno se statistickými údaji o grafu jako je počet uzlů, počet hran, průměr a hustota grafu a další viz obrázek 16



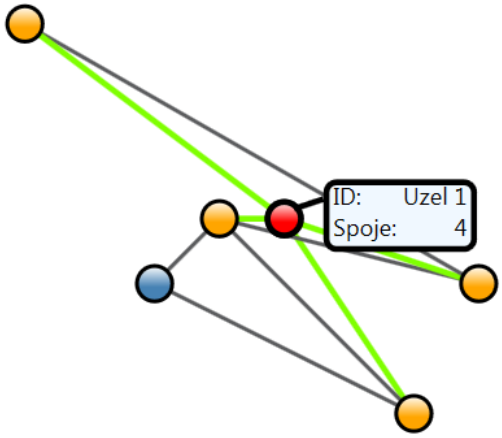
Obrázek 16: Sekundární okno se statistickými údaji

Zobrazit názvy uzlů
Ačkoliv jsou názvy uzlů zobrazovány automaticky při přejetím kurzoru myši přes uzel, může si uživatel pomocí položky `Zobrazit názvy uzlů` z kategorie `Zobrazení` pro lepší orientaci v grafu zapnout nebo vypnout zobrazení názvů všech uzlů. Pro označený uzel stále platí, že se zobrazuje i údaj o počtu incidentních hran viz obrázek 17



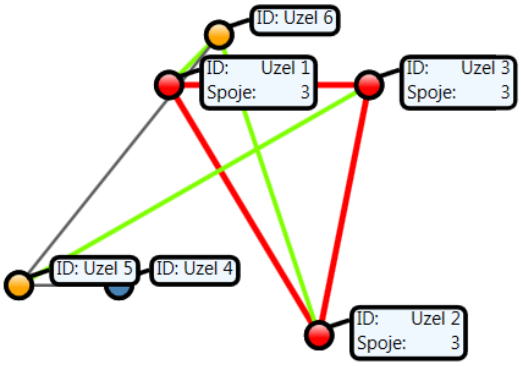
Obrázek 17: Zapnuto zobrazování názvů

Zobrazovat sousední uzly
V kategorii `Zobrazení` je také položka `Zobrazovat sousedníuzly`. Při zapnutí této možnosti budou s vybraným označeným uzlem zároveň označeny i jeho sousední uzly a hrany mezi nimi.



Obrázek 18: Zapnuto zobrazování sousedních uzlů

Výběr více uzlů na jednou
Aplikace `Nettop` umožňuje v grafu označit více uzlů najednou. Toto může uživatel provést, pokud bude při klikání na jednotlivé uzly držet stisknutou klávesu `Ctrl`. Navíc, pokud uživatel označí více uzlů, které jsou navzájem spojeny hranou, jsou tyto hrany vyznačeny červeně. Toto umožňuje uživateli přehledně vyznačovat celé cesty mezi dvěma a více uzly. Při označování více uzlů je samozřejmě možné používat jak možnosti `Zobrazovat sousedni uzly` tak i `Zobrazit názvy uzlů` jak ukazuje obrázek 19



Obrázek 19: Ukázka výběru více uzlů

5 Realizované modely topologie

V této kapitole budou popsány modely topologií, které byly realizovány v rámci praktické části bakalářské práce. Součástí popisu jednotlivých modelů jsou i části zdrojového kódu aplikace tak, aby byl zřejmý způsob jejich realizace.

5.1 Model Waxman

Jedná se o jeden z nejstarších modelů navržených B. M. Waxmanem [7]. Tento model generuje náhodný neorientovaný graf, který může reprezentovat například inter-doménové spoje. Tento model se nezabývá hierarchií a strukturou sítě Internet, ale rozmístěním jednotlivých uzlů a vztahy (spojeními) mezi nimi.
Výhodou tohoto algoritmu je jeho jednoduchost a rychlost avšak grafy generované tímto modelem nesledují mocninné zákony. Tento model byl později překonán v přesnosti s jakou zachycuje topologii rozsáhlých sítí novějšími modely, které jsou založené přímo na mocninných zákonech. Stále je však vhodné jej používat pro generování topologií menších sítí nebo jako součást hierarchických modelů, ve kterých může mít na starost generování topologie routerů uvnitř AS.

5.1.1 Popis algoritmu

Nejprve je vygenerován seznam uzlů a ty jsou náhodně rozmístěny na 2D rovině. Poté jsou do grafu umístěny hrany. Pravděpodobnost jestli jsou dva uzly spojeny hranou je dána funkcí:

$$P(u, v) = \beta \times e^{\frac{-d(u,v)}{L\alpha}} \tag{6}$$

kde:

u, v	= vybrané uzly
β	= hodnota intervalu $\langle 0,1\rangle$, čím vyšší, tím větší počet hran v grafu
α	= hodnota intervalu $\langle 0,1\rangle$, čím vyšší, tím větší poměr delších hran ke kratším
$-d(u, v)$	= Euklidovská vzdálenost mezi vybrané uzly
L	= největší vzdálenost mezi dvěma uzly v grafu

Model je dále popsán pomocí pseudokódu:

1: uzly $\leftarrow$ počet uzlů z parametru metody
2: P //pravděpodobnostní funkce 6
3: $G = (V, E)$ //graf sítě, V je množina uzlů, E je množina hran grafu
4: **for** $i = 0$ to $uzly$ **do**
5: náhodně umísti uzel V[i]
6: **end for**
7: **for all** (u, v) takové, že $u, v \in V; (u, v) \notin E; u \neq v$ **do**
8: **if** $P(u, v)$ **then**
9: $E \leftarrow (u, v)$
10: **end if**
11: **end for**
12: return $G = (V, E)$

Pseudokód 1: Waxman model

5.1.2 Popis implementace

Generování grafu podle modelu Waxman má na starosti metoda `RunModel` třídy `ModelWaxman`, kterou ukazuje zdrojový kód 8. Uživatel může ovlivnit pouze některé argumenty této metody a to počet uzlů v grafu a koeficienty α a β.

```
1 public NetworkViewModel RunModel(
2 BackgroundWorker worker,
3 int nodes, // počet uzlů v grafu
4 int xmin, int xmax, int ymin, int ymax, // rozměry plochy, na které bude graf
      umístěn
5 float alfa, float beta, // nastavení alfa a beta koeficientů
6 double nodeSpan) // minimální vzdálenost mezi uzly
```

Zdrojový kód 8: Metoda ModelWaxman.RunModel

Velikost plochy určené pro graf je definována automaticky metodou `RunModel`. Plocha je následně rozdělena na mřížku tak, aby se na plochu přehledně vešly všechny uzly a nepřekrývaly se. Jednotlivým uzlům jsou pak náhodně přidělovány pozice v mřížce tak, aby byla dodržena podmínka jejich náhodného rozmístění. Toto ukazuje zdrojový kód 9

Implementace modelu Waxman obsahuje další větev programu pro výpočet plochy a rozmístění uzlů modelu Transit-Stub. Hlavní rozdíl je v tom, že model Transit-Stub si sám určuje rozměry jednotlivých domén přes vstupní parametry metody `RunModel`. Další rozdíl je v tom, že koncové neboli Stub domény jsou kolem tranzitních domén rozmisťovány v kruhu a ne do mřížky.

```
...
 int lX = 0;
 // nastavení rozměrů plochy; jde o součet velikosti uzlu, celkovou
     plochu všech uzlů dohromady a mezer mezi uzly
int hX = (int)((2 * nodeSize) + (nodeSize * nodes) + (nodes * 10));
int lY = 0;
int hY = (int)((2 * nodeSize) + (nodeSize * nodes) + (nodes * 10));

//vytvoření mřížky pro umisťování uzlů
//mřížka je odsazena od kraje plochy o velikost uzlu, aby uzly
    nepřesahovaly přes okraj
double gridStartX = lX + nodeSize;
double gridStartY = lY + nodeSize;
//plocha volná k umístění uzlů je celková plocha - okraje
double availableSpaceX = hX - (2 * gridStartX);
double availableSpaceY = hY - (2 * gridStartY);

//výpočet kolik plochy zabere umístění jednoho uzlu
double nodeRadius = nodeSize + (hX + hY) / 200;

//rozdělení volné plochy na mřížku
double linesX = availableSpaceX / nodeRadius;
double linesY = availableSpaceY / nodeRadius;

// vytvoření seznamu pozic pro umístění uzlu
suitablePositions.Clear();
for (int i = 0; i < linesX; i++)
{
        for (int j = 0; j < linesY; j++)
        {
                suitablePositions.Add(new Point(gridStartX + i *
                    nodeRadius, gridStartY + j * nodeRadius));
        }
}
// vytvoření uzlu a jeho náhodné umístění do mřížky
for (int nodeId = 0; nodeId < nodes; nodeId++)
{
        selectedPos = suitablePositions[r.Next(suitablePositions.Count)];
        newNode = new NodeViewModel(selectedPos);
        newNode.Name = "Uzel " + (network.Nodes.Count + 1);
        network.Nodes.Add(newNode);
        suitablePositions.Remove(selectedPos);
}
...
```

Zdrojový kód 9: Rozmisťování uzlů

Zdrojový kód 10 ukazuje, že metoda `RunModel` dále projde ve dvou vnořených cyklech všechny kombinace uzlů a pokud daná dvojce vyhovuje podmínkám je mezi tyto dva uzly přidána hrana. Návratovou hodnotou metody `RunModel` je graf sítě, který je vrácen uživatelskému rozhraní a zobrazen uživateli.

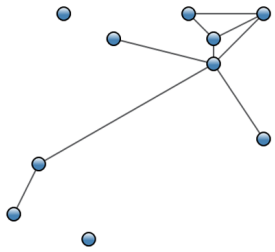
```
...
//dva vnořené cykly, které projdou všechny uzly
for (int firstNode = 0; firstNode < nodes; firstNode++)
{
        for (int secondNode = 0; secondNode < nodes; secondNode++)
        {
                ...

                bool containsAlready = false;
                // kontrola jestli nejde o stejný uzel
                if (firstNode == secondNode) continue;
                // kontrola jestli oba uzly již nejsou spojeny
                foreach (ConnectionViewModel c in network.Connections)
```

```
14                          {
15                                  if (c.SourceConnector.ParentNode ==
                                        network.Nodes[secondNode] &&
                                        c.DestConnector.ParentNode ==
                                        network.Nodes[firstNode]) containsAlready =
                                        true;
16                          }
17
18                          if (!containsAlready)
19                          {
20                                  // výpočet vzdálensoti obou uzlů od sebe
21                                  double dist = ...
22                                  //výpočet pravděpodobnosti
23                                  double p = (beta * Math.Exp(-dist / (alfa *
                                        maxDist)));
24
25                                  if (r.NextDouble() < p)
26                                  {
27                                          ...
28                                          network.Connections.Add(connection);
29                                  }
30                          }
31
32                  }
33          }
34          ...
35          //návratová hodnota
36          return network;
```

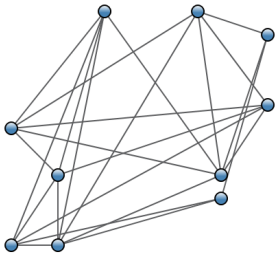
Zdrojový kód 10: Spojování uzlů hranami

5.1.3 Charakteristiky grafu

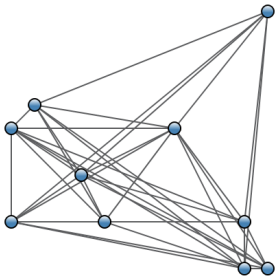
Waxmanův model generuje náhodné grafy, jejichž souvislost je ovlivněna parametry α a β jeho pravděpodobností funkce. Z implementace pravděpodobností funkce a z obrázků 20 až 22 vyplývá, že vstupní parametr α ovlivňuje délku hran mezi uzly a parametr β ovlivňuje počet rhan v grafu. V závislosti na vypočtených hodnotách pravděpodobnostní funkce může být výsledný graf i kompletní. Toto však předpokládá hodnoty parametrů α a β blížící se jedné.



Obrázek 20: 10 uzlů, $\alpha = 0.2$, $\beta = 0.5$



Obrázek 21: 10 uzlů, $\alpha = 1.0$, $\beta = 0.5$



Obrázek 22: 10 uzlů, $\alpha = 1.0$, $\beta = 1.0$

Z obrázku 20 je patrná nespojitost grafu při menších hodnotách obou parametrů, na obrázku 21 je vidět, že zvýšením hodnoty parametru α dosáhneme grafu s vyšším počtem dlouhých hran tak jak to popisuje teorie [7]. Obrázek 22 ukazuje, že i když zvýšením β dosáhneme větší hustoty grafu, nemusí být výsledný graf stále kompletní.

5.2 Model Barabási–Albert

Model se pokouší simulovat topologii sítě Internet tak, že se snaží zachytit její postupný růst. Grafy generované tímto modelem částečně sledují mocninné zákony. Tento model postupně přidává do grafu nové uzly, které se snaží připojit spíše k uzlům, které již mají větší počet spojení, aby si tak zajistily snadnější přístup do zbytku sítě přes co nejmenší počet uzlů. V grafu tak postupně vznikají centra, uzly s velkými počty spojení, což je jedna z charakteristik sítě Internet.

5.2.1 Popis algoritmu

Slovní popis algoritmu by se podle [13] [2] dal rozdělit na dva úkony. Nejprve je podle vstupních parametrů vygenerován základní graf. Takový graf je neorientovaný a spojitý. Do tohoto grafu jsou postupně přidávány uzly dokud počet uzlů v grafu nedosáhne požadovaného počtu. Každý přidaný uzel je spojen s dalšími pomocí funkce

$$P(v) = d_v / \sum_j d_j \tag{7}$$

kde:

d_v = stupeň uzlu v
j = aktuální počet uzlů v grafu
$\sum_j d_j$ = suma všech stupňů uzlů

Z výše definice je patrná snaha připojovat nově přidané uzly s uzly s vyšším stupněm a tedy větším počtem spojů.

Z dostupné odborné literatury [7] [2] jsem sestavil pseudokód algoritmu:

1: $G = (V, E)$ //graf sítě, V je množina konečných uzlů, E množina hran
2: P //pravděpodobnostní funkce 7
3:
4: $s \leftarrow$ počáteční počet uzlů z parametru metody
5: $n \leftarrow$ konečný počet uzlů z parametru metody
6: $d \leftarrow$ stupeň přidávaných uzlů z parametru metody
7: $|E| \leftarrow 0$
8:
9: // vytvoření kompletního grafu
10: **for** $i = 0$ to s **do**
11: Vytvoř u a $V \leftarrow u$
12: **for all** $v \in V$ takové, že $u, v \in V; (u, v) \notin E; u \neq v$ **do**
13: $E \leftarrow (u, v)$
14: **end for**
15: **end for**
16:
17: //doplnění zbývajících uzlů
18: **for** $i = 0$ to $(n - s)$ **do**
19: vytvoř u a $V \leftarrow u$
20: **for** $j = 0$ to d **do**
21: **for all** (u, v) takové, že $u, v \in V; (u, v) \notin E; u \neq v$; P(u,v) **do**
22: $E \leftarrow (u, v)$
23: **end for**
24: **end for**
25: **end for**
26: return $G = (V, E)$

Pseudokód 2: Barabási-Albert model

5.2.2 Popis implementace

Zde, podobně jako u Waxmanova modelu, je generování grafu prováděno metodou `RunModel` třídy `ModelBA`. Zdrojový kód 11 ukazuje, že vstupní parametry modelu jsou tři a to základní a koncový počet uzlů a maximální stupeň uzlu, tedy počet spojů obsahujících daný uzel.
Maximální hodnoty základního počtu a maximálního stupně uzlů jsou omezeny nastavením ostatních parametrů. Rozsahy vstupních paramterů ukazuje následující tabulka:

Vstupní parametr	Min. hodnota		Max. hodnota
základní počet uzlů	2	až	100
konečný počet uzlů	1	až	aktuální základní počet uzlů
maximální stupeň uzlu	1	až	aktuální max. stupeň uzlu

Tabulka 1: Rozsahy vstupních parametrů modelu Barabási-Albert

```
1  public NetworkViewModel RunModel
2  (BackgroundWorker worker, //vlákno na kterém metoda běží
3  int initNodes, // počet uzlů základního grafu
4  int totalNodes, // konečný počet uzlů
5  int degree) //stupeň připojovaných uzlů
```

Zdrojový kód 11: Metoda ModelBA.RunModel

Plocha potřebná pro rozmístění uzlů, je opět vypočítána automaticky podle vstupních parametrů podobně jako u modelu Waxman viz zdrojový kód 9.

Po vytvoření plochy pro umístění grafu je tedy vygenerován počáteční spojitý graf. Při spojování uzlů grafu udržujeme v poli `nodeDegrees` záznam o stupních jednotlivých uzlů abychom nepřekročili limit definovaný vstupním parametrem maximálního stupně.

```
1 //Pole, které udržuje záznam o stupních jednotlivých uzlů
2 int[] nodeDegrees = new int [totalNodes];
3 //Základní spojitý graf
4 for (int i = 0; i < initNodes; i++)
5 {
6         for (int j = 0; j < initNodes; j++)
7         {
8                 //přeskočit pokud jsou uzly stejné
9                 if (i == j) continue;
10
11                //testuje pokud již spojení mezi těmito uzly neexistuje
12                foreach (ConnectionViewModel c in network.Connections)
13                {
14                    if (c.SourceConnector.ParentNode == network.Nodes[j] &&
                          c.DestConnector.ParentNode == network.Nodes[i])
                          containsAlready = true;
15                }
16                if (containsAlready) continue;
17                //Vytvoří novou hranu
18                network.Connections.Add (new
                      ConnectionViewModel(network.Nodes[i], network.Nodes[j],
                      ConnectionViewModel.ConnectionType.Unoriented));
19                ...
20                //Zvýšíme stupně obou spojených uzlů
21                nodeDegrees[i]++;
22                nodeDegrees[j]++;
23        }
24 }
```

Zdrojový kód 12: Vytvoření základního grafu

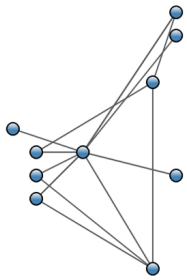
Následně ve třech vnořených cyklech nejprve přidáme do grafu nový uzel a poté jej spojíme příslušným počtem hran s ostatními uzly v grafu. Při spojování uzlů procházíme všechny uzly grafu a po nalezení prvního vyhovujícího uzlu přerušíme aktuální iteraci smyčky. Tím zajistíme, že pro každou hranu hledáme dvojici uzlů s různým výsledkem pravděpodobnostní funkce.

```
1  //Přidáváme uzly dokud nedosáhneme konečného počtu uzlů
2  for (int firstNode = initNodes; firstNode < totalNodes; firstNode++)
3  {
4          ...
5          //přidání uzlu do grafu
6          selectedPos = suitablePositions[r.Next(suitablePositions.Count)];
7          newNode = new NodeViewModel(selectedPos);
8          newNode.Name = "Uzel " + (network.Nodes.Count + 1);
9          network.Nodes.Add(newNode);
10         suitablePositions.Remove(selectedPos);
11
12         //přidáme počet hran podle vstupního parametru stupně
13         for (int m = 0; m < degree; m++)
14         {
15                 //projdeme všechny stávající uzly a pokusíme se je spojit s tím,
                       který jsme vytvořili
16                 for (int secondNode = 0; secondNode < firstNode; secondNode++)
17                 {
18                         containsAlready = false;
19                         // kontrola jestli nejde o stejný uzel
20                         if (firstNode == secondNode) continue;
21                         // kontrola jestli spojení mezi těmito uzly již neexistuje
22                         foreach (ConnectionViewModel c in network.Connections)
23                         {
24                                 ...
25                         }
26
27                         if (!containsAlready)
28                         {
29                                 //pravděpodobnostní funkce:
30                                 double p =
                                       (double)((double)nodeDegrees[secondNode]) /
                                       (double)(2.0d * network.Connections.Count);
31                                 //pokud projdeme funkcí, spojíme uzly
32                                 if (p > r.NextDouble())
33                                 {
34                                         //vytvoř hranu mezi uzly i a j
35                                         network.Connections.Add(new
                                               ConnectionViewModel(firstNode,
                                               secondNode,
                                               ConnectionType.Unoriented));
36                                         //zvýšíme stupně obou spojených uzlů
37                                         nodeDegrees[firstNode]++;
38                                         nodeDegrees[secondNode]++;
39                                         //jakmile najdeme spojení, přerušíme
                                               stávající smyčku
40                                         break;
41                                 }
42                         }
43                 }
44         }
45 }
46 ...
47 //návratová hodnota
48 return network;
```

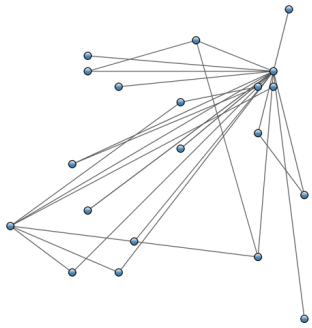
Zdrojový kód 13: Připojení nových uzlů

5.2.3 Charakteristiky grafu

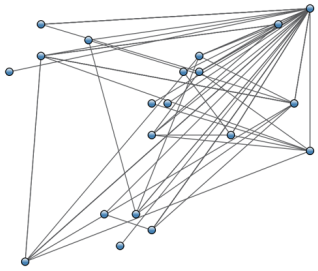
Z grafů vygenerovaných pomocí modelu Barabasi-Albert jsou už na první pohled vidět centrální uzly, které přitahují spojení s ostatními uzly. Při menším konečném počtu uzlů mají grafy jeden až dva uzly s více spojeními jako na obrázku 23. S větším konečným počtem uzlů poté narůstá i větší počet centrálních uzlů. Obrázek 25 ukazuje, že při zvýšení maximálního stupně nedochází ke vzniku většího počtu centrálních uzlů ale spíše k ještě většímu růstu stupně těch aktuálních.



Obrázek 23: 10 uzlů, základ = 3, $d = 3$



Obrázek 24: 20 uzlů, základ = 10, $d = 3$



Obrázek 25: 20 uzlů, základ = 10, $d = 10$

5.3 Model Power Law Out Degree

Power Law Out Degree model neboli PLOD, je abstraktní model, u kterého je generování grafu řízeno mocninným zákonem stupně. Vlastnosti vygenerovaných grafů tedy sledují mocninné zákony, a tudíž lze předpokládat, že i do jisté míry odpovídají vlastnostem reálných počítačových sítí. PLOD model bývá využíván zejména pro generování topologií na úrovni autonomních systémů, typicky sítě Internet.

5.3.1 Popis algoritmu

Protože algoritmus vychází z mocninného zákona stupně, je nejprve uzlům v grafu přiřazen tzv. kredit stupně. Rozvržení kreditu odpovídá exponenciálnímu rozdělení

$$\beta x^{-\alpha} \tag{8}$$

kde:

x = hodnota 1...(|V|-1); |V| je počet uzlů v grafu
β = parametr exponenciálního rozdělení
α = parametr exponenciálního rozdělení

To znamená, že pokud by byly uzly seřazeny sestupně podle velikosti stupně uzlu, bude uzel s pořadím x mít stupeň uzlu roven $\beta x^{-\alpha}$.

Dvojice uzlů jsou poté ve smyčce spojovány hranami tak, aby výsledný stupeň uzlu odpovídal kreditu, který mu byl na začátku generování grafu přiřazen. Dvojice uzlů, které by měly být spojeny jsou vybírány náhodně. Každému z dvojice uzlů, které jsou spojeny hranou je zároveň snížen jeho kredit. Jakmile má uzel kredit roven nule, nelze jej spojit s jiným uzlem. Takto postupujeme, až dokud nám nezůstanou žádné vhodné dvojce uzlů ke spojení hranou. Poté je generování grafu ukončeno.

1: uzly ← počet uzlů z parametru metody
2: α, β //vstupní parametry metody
3: $G = (V, E)$ //graf sítě, V je množina uzlů, E je množina hran
4: **for** $i = 0$ to $uzly$ **do**
5: u = random(1, uzly)
6: $deg_u = \beta x^{-\alpha}$
7: $V \leftarrow u$
8: **end for**
9: **while do**
10: $u = random(1, |V|)$
11: $v = random(1, |V|); v \neq u$
12: **if** $deg_u > 0$ AND $deg_v > 0$ AND $(u, v) \notin E$ **then**
13: $E \leftarrow (u, v)$
14: $deg_u - -$
15: $deg_v - -$
16: **end if**
17: **end while**
18: return $G = (V, E)$

Pseudokód 3: PLOD model

5.3.2 Popis implementace

Metoda RunModel třídy ModelPLOD, kterou ukazuje zdrojový kód 14, využívá ke generování grafu tři vstupní parametry a sice α, β a počet uzlů v grafu. Pomocí nastavení parametrů α a β můžeme dosáhnout námi požadovaného počtu hran v grafu. Zvyšování parametru β ovlivňuje průměrný stupeň uzlů grafu. Hodnota parametru α naopak ovlivňuje pravděpodobnost, že budou dva uzly spojeny hranou. Z rovnice 8 nám tedy vyplývá, že s vyšší hodnotou α se snižuje hustota výsledného grafu.

```
1 public NetworkViewModel RunModel
2 (BackgroundWorker worker, //vlákno, na kterém metoda běží
3 int nodes, //počet uzlů v grafu
4 double alpha, //vstupní parametr ovlivňující distribuci stupně
5 double beta) //vstupní parametr ovlivňující distribuci stupně
```

Zdrojový kód 14: Metoda ModelPLOD.RunModel

Generování grafu začíná výpočtem plochy a rozměrů grafu. Plocha je čtvercová a jako u předchozích modelů je její velikost vypočtena převážně z velikosti všech uzlů, vzdáleností mezi nimi a okrajů plochy. Dále pokračujeme vytvořením seznamu uzlů a přiřazením jejich kreditu. V tuto chvíli je také zadefinované pole `nodeCredit` které udržuje informaci o stavu kreditu jednotlivých uzlů a seznam polí `nodeMatrix` se všemi kombinacemi uzlů, které je možné spojit hranou.

```
1  //pole udržující informaci o stavu kreditů všech uzlů
2  int[] nodeCredit = new int [nodes];
3  ...
4  //pro sledování volných kombinací spojení využijeme dvourozměrné pole
5  List<int[]> nodeMatrix = new List<int[]>();
6  for (int i = 0; i < nodes; i++)
7  {
8          for (int j = 0; j < nodes; j++)
9          {
10                 if (nodeMatrix.Contains(new int[]{j, i})) continue;
11                 if (i != j) nodeMatrix.Add(new int[] {i, j});
12                 network.L[i, j] = 0.0;
13         }
14 }
15
16 ...
17 //V této části probíhá výpočet plochy grafu, je principiálně stejný jako u
       předchozích modelů
18 //Je však o dost jednodušší, protože závisí pouze na počtu uzlů a na jejich
       velikosti
19 ...
20
21 //vygenerování seznamu uzlů
22 for (int nodeId = 0; nodeId < nodes; nodeId++)
23 {
24         selectedPos = suitablePositions[r.Next(suitablePositions.Count)];
25         newNode = new NodeViewModel(selectedPos);
26         newNode.Name = "Uzel " + (network.Nodes.Count + 1);
27         network.Nodes.Add(newNode);
28         suitablePositions.Remove(selectedPos);
29 }
30
31 //výpočet kreditů jednotlivých uzlů podle exponenciálního rozdělení
32 for (int i=0; i<nodes; i++)
33 {
34         int x = r.Next(1, nodes+1);
35         double pow = Math.Pow(x, (-alpha));
36         nodeCredit[i] = (int)Math.Round((beta *
               pow),MidpointRounding.AwayFromZero);
37 }
```

Zdrojový kód 15: Vytvoření seznamu uzlů a distribuce stupně uzlů

Zdrojový kód 16 popisuje spojování uzlů hranami. V jedné `while` smyčce je nejdříve náhodně vybrána dvojice uzlů, které chceme spojit. Stejně tak je v seznamu dvojic nalezena i její opačná varianta. Důvodem hledání opačné dvojice je, že v neorientovaném grafu bychom v případě spojení těchto uzlů hranou akorát zdvojovali jedno a to samé spojení. Proto jsou při spojení hranou ze seznamu vymazány obě dvojice.

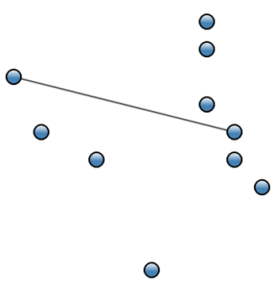
Samotné spojení dvou uzlů hranou je podmíněno pouze tím, že jejich kredit musí být větší než 0. Často zmiňovaným nedostatkem tohoto modelu bývá, že na konci generování grafu nám mohou zůstat uzly s nenulovým kreditem, které již nelze spojit s žádným jiným uzlem. Tento nedostatek se pokouší odstranit třeba modifikace tohoto modelu s názvem PLDA [12]

```
1 while (doLoop)
2 {
3         //náhodně vybereme dvojici uzlů, kterou bychom chtěli spojit
4         int combinationIndex = r.Next(0, nodeMatrix.Count);
5         int[] combination = nodeMatrix[combinationIndex];
6         int[] combinationReverse = null;
7         int firstNode = combination[0];
8         int secondNode = combination[1];
9         //ke zvolené dvojci najdeme také opačnou dvojici
10        foreach (int[] comb in nodeMatrix)
11        {
12                if (comb[0] == secondNode && comb[1] == firstNode)
                      combinationReverse = comb;
13        }
14        //pokud mají oba uzly kredit větší než 0, spojíme je hranou
15        if (nodeCredit[firstNode] > 0 && nodeCredit[secondNode] > 0)
16        {
17                //vytvoření nové hrany
18                network.Connections.Add(new
                      ConnectionViewModel(network.Nodes[firstNode],
                      network.Nodes[secondNode],
                      ConnectionViewModel.ConnectionType.Unoriented));
19                ...
20                //dvojice odebereme ze seznamu volných dvojic, odstraníme i
                      opačnou dvojici, protože graf je neorientovaný
21                nodeMatrix.Remove(combination);
22                nodeMatrix.Remove(combinationReverse);
23
24                //oběma uzlům snížíme kredit, protože byly spojeny hranou
25                nodeCredit[firstNode]--;
26                nodeCredit[secondNode]--;
27        }
28
29        //kontrola jestli je možná ještě nějaká varianta spojení
30        ...
31        if (creditSum <= 1 || nodeMatrix.Count == 0 || noConnectionsPossible ==
              nodeMatrix.Count) doLoop = false;
32 }
33 ...
34 // návratová hodnota
35 return network;
```

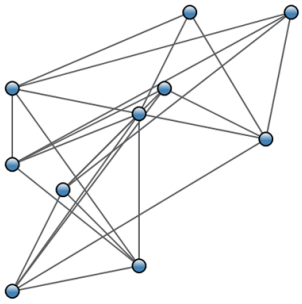
Zdrojový kód 16: Smyčka spojující uzly hranami

5.3.3 Charakteristiky grafu

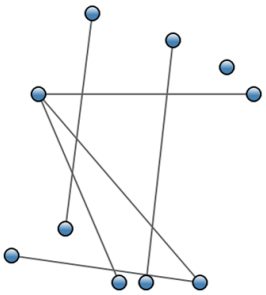
Výsledné grafy reflektují vlastnosti popsané v [10] a [2]. Odpovídá i chování vstupních parametrů α a β. Z obrázků 26 až 28 je patrné jak postupné zvyšování parametru β zvyšuje počet hran a tím i průměrný stupeň uzlu v grafu. Naopak zvyšování parametru α na obrázku 28 snižuje hustotu grafu i při vyšších hodnotách β. V těchto případech nejsou výjimkou ani velmi řídké grafy. Tabulka 2 ukazuje údaje grafů na obrázcích včetně průměrného stupně grafu k a hustoty grafu D.



Obrázek 26: 10 uzlů, $\alpha = 3$, $\beta = 3$



Obrázek 27: 20 uzlů, $\alpha = 10$, $\beta = 3$



Obrázek 28: 20 uzlů, $\alpha = 10$, $\beta = 10$

Obrázek	**n**	α	β	D	k
Obr. č. 26	10	0.10	1.00	0.022	0
Obr. č. 27	10	0.10	6.00	0.511	4
Obr. č. 28	10	1.10	6.00	0.133	1

Tabulka 2: Údaje vzorových grafů

5.4 Model Transit-Stub

Model Transit-Stub je zástupcem hierarchických modelů topologie. Zachycuje strukturu sítě a její rozvržení do vlastních autonomních systémů neboli domén. Stejně tak se model věnuje vzájemnému propojení těchto autonomních systémů. Výsledné grafy sestávají z tranzitních a koncových domén. Zatímco tranzitní domény tvoří formu páteřní sítě umožňující přístup do zbytku sítě, jedna nebo více koncových domén jsou připojeny k routerům v tranzitních doménách. Toto může simulovat lokální poskytovatele internetového připojení, kteří poskytují své služby domácím nebo firemním LAN sítím.

5.4.1 Popis algoritmu

Podle algoritmu modelu Transit-Stub je nejprve vygenerován spojitý neorientovaný graf. Tento graf reprezentuje tranzitní síť na úrovni domén, tedy jeden uzel grafu se rovná jedné tranzitní doméně. Následně je každý uzel v tomto grafu nahrazen vlastním spojitým neorientovaným grafem a ty jsou podle vzoru předchozího grafu pospojovány mezi sebou.

Každá tranzitní doména je tedy nyní zastoupena několika routery, které umožňují komunikaci v rámci sítě a připojení koncových domén. Dalším krokem je, že pro každý router ve všech tranzitních doménách jsou vygenerovány grafy koncových domén a ty jsou s nimi následně pospojovány. Posledním krokem je přidání nadbytečných spojů mezi několik náhodně vybraných dvojic tranzitní - koncová a koncová - koncová doména, aby byla simulována i záložní spojení pro případy technických problémů některých domén.

1: $G = (V, E)$ //graf sítě, V je množina uzlů, E je množina hran
2: $T = (M, F)$ //graf pro pomocné výpočty, M je množina uzlů, F je množina hran
3: $tD \leftarrow$ počet 'Transit'domén z parametru metody
4: $tN \leftarrow$ počet 'Transit'uzlů z parametru metody
5: $sD \leftarrow$ počet 'Stub' domén na jeden tranzitní uzel z parametru metody
6: $sN \leftarrow$ počet 'Stub' uzlů z parametru metody
7:
8: // vytvoření spojitého grafu tranzitních domén
9: **for** $i = 0$ to tD **do**
10: Vytvoř u a $V \leftarrow u$
11: **for all** $v \in V$ takové, že $u, v \in N; (u, v) \notin E; u \neq v$ **do**
12: $E \leftarrow (u, v)$
13: **end for**
14: **end for**
15:
16: //doplnění grafu tranzitními uzly
17: **for all** $v \in V$ **do**
18: Generuj $T(M, F); |M| = tN;$
19: $V \leftarrow M; E \leftarrow F$
20: **end for**
21:
22: //vygenerování stub domén
23: **for** $i = 0$ to $(tD * tN)$ **do**
24: **for** $j = 0$ to sD **do**
25: Generuj $T(M, F); |M| = sN;$
26: Vyber u, v takové že u je tranzitní uzel; $v \in M; (u, v) \notin E;$
27: $E \leftarrow (u, v)$
28: $V \leftarrow M; N \leftarrow F$
29: break;
30: **end for**
31: **end for**
32: return $G = (V, E)$

Pseudokód 4: Trasnit-Stub model

5.4.2 Popis implementace

S ohledem na algoritmus modelu, přijímá metoda ModelTransitStub.RunModel čtyři vstupní parametry, které ovlivňují výsledný graf. Zatímco počet tranzitních domén je pevně daný vstupním parametrem, další tři vstupní parametry jsou průměrné hodnoty. Při implementaci jsou tedy tyto hodnoty voleny náhodně za dodržení celkového průměru. Tato náhodnost nám zajistí, že se domény budou od sebe více odlišovat a více odrážet realitu, kde se dvě stejné domény velmi často nevyskytují.

Vzhledem k vzájemné závislosti jednotlivých parametrů, může výrazné zvýšení některého z nich značně prodloužit i čas potřebný k vygenerování grafu. Z tohoto důvodu byly horní hranice těchto parametrů v aplikaci Nettop nastaveny tak, aby dovolovaly generovat grafy v čase přijatelném z hlediska názornosti.

```
1 public NetworkViewModel RunModel
2 (BackgroundWorker worker,  // vlákno, na kterém metoda běží
3  int transitDomains, // počet tranzitních domén
4  int avgTransitNodes, // průměrný počet tranzitních uzlů v jedné tranzitní doméně
5  int avgStubDomains, // průměrný počet koncových domén na jeden tranzitní uzel
6   int avgStubNodes) // průměrný počet uzlů v jedné koncové doméně
```

Zdrojový kód 17: Metoda ModelTransitStub.RunModel

Aby grafy vygenerované podle tohoto modelu byly co nejpřehlednější, bylo třeba řádně promyslet způsob rozmisťování jednotlivých domén. Právě s ohledem na přehlednost je velikost plochy, mřížka pro umisťování domén a tím i velikost grafu definována počtem tranzitních a koncových domén v grafu. Zvolené parametry, které ukazuje zdrojový kód 18, tak umožňují přehledné rozmístění i grafů s větším počtem domén a tedy i uzlů.

```
1 double desiredTransitSize = 200; // průměr tranzitní domény
2 double desiredStubSize = desiredTransitSize * 0.25; // průměr koncové domény
3 // vzdálenost koncových domén od tranzitních, zvyšuje se s počtem domén
4 radius = avgStubDomains * avgStubNodes * (int)desiredStubSize;
5 ...
6 int xmin = 0;
7 int xmax = (int)
8 (2 * (radius + desiredTransitSize + desiredStubSize) + // okraje plochy
9 (transitDomains * desiredTransitSize) + // plocha tranzitních domén
10 (avgStubDomains * desiredStubSize * avgTransitNodes) + // plocha koncových domén
11 (2 * transitDomains * radius)); //mezery mezi doménami
12 int ymin = 0;
13 int ymax = (int)(2 * (radius + desiredTransitSize + desiredStubSize) +
       (transitDomains * desiredTransitSize) + (avgStubDomains * desiredStubSize *
       avgTransitNodes) + (2 * transitDomains * radius));
```

Zdrojový kód 18: Definice rozměrů grafu

Jak je uvedeno v podkapitole 5.4.1, nejprve by měl být vytvořen náhodný graf definující tranzitní domény a následně by měl být každý uzel tohoto grafu nahrazen vlastním náhodným grafem. Mnou implementovaný model Transit-Stub využívá pro tyto grafy model Waxman. To mi dovolilo využít část jeho kódu k tomu, abych mohl rovnou generovat celé tranzitní domény a rovnou je vzájemně spojit a umístit do grafu viz zdrojový kód 19. Spojování tranzitních domén probíhá pomocí pravděpodobnostní funkce stejně jako u modelu Waxman.

```
1 //smyčka generující tranzitní domény
2 for (int i = 0; i < transitDomains; i++)
3 {
4         domPos = domainPositions[r.Next(domainPositions.Count)]; //náhodný výběr
              pozice domény
5         lX = domPos.X - desiredTransitSize/4; //definice rozměrů domény
6         hX = domPos.X + desiredTransitSize/4; //definice rozměrů domény
7         lY = domPos.Y - desiredTransitSize/4; //definice rozměrů domény
8         hY = domPos.Y + desiredTransitSize/4; //definice rozměrů domény
9         nodes = r.Next(avgTransitNodes); // náhodý počet uzlů v doméně
10        if (nodes <= 0) nodes = 1; //počet je alespoň 1
11
12        tempNetwork = new NetworkViewModel(); // pomocný graf
13        waxman.TransitNumber = i + 1; //číslo tranzitní domény
14        //vygenerování tranzitní domény pomocí modelu Waxman
15        tempNetwork = waxman.RunModel(worker, avgTransitNodes, (int)lX, (int)hX,
              (int)lY, (int)hY, alfa, beta, desiredTransitSize/2);
16        tempNetwork.CountNetworkMetrics();
17        transitDomainsList.Add(tempNetwork); //seznam tranzitních domén
18        ...
19        //pomocný graf `zkopírujeme` do výsledného grafu
20        completeNetwork.Nodes.AddRange(tempNetwork.Nodes);
21        completeNetwork.Connections.AddRange(tempNetwork.Connections);
22        completeNetwork.NodeGroups.Add(domain);
23
24        domainPositions.Remove(domPos); //pozici domény odstrníme ze seznamu
              volných pozic
25        transitDomainPositions.Add(domPos); //udržujeme záznam o pozicích domén
26
27 }
```

Zdrojový kód 19: Vytvoření tranzitních domén

Po umístění tranzitních domén, následuje vygenerování koncových domén. Koncové domény jsou rozmístěny v kruhu kolem tranzitní domény, ke které jsou připojeny. Tato část zdrojového kódu je velmi podobná generování tranzitních domén s tím rozdílem, že je umístěna do více vnořených cyklů. Vnořených cyklů je zde využito kvůli tomu, aby bylo mohlo být zabezpečeno vygenerování a rozmístění grafů všech koncových domén pro každý tranzitní uzel v grafu.

```
1 //definice počtu pozic okolo tranzitní domény a 'kroku' pro rozmist'ování
      koncových domén
2 int avgDomainPos = avgTransitNodes * avgStubDomains;
3 double step = 360 / avgDomainPos;
4
5 for (int t=0; t < transitDomains; t++)
6 {
7         ...
8         rndNetwork = transitDomainsList[t]; //výběr tranzitní domény ze seznamu
9
10        //vlastní pozice koncových domén; jsou umístěny v kruhu
11        for (int i = 0; i < avgDomainPos; i++)
12        {
13                xPos = networkCenter.X + ((radius + desiredTransitSize/2) *
                      Math.Cos(angle * Math.PI / 180));
14                yPos = networkCenter.Y + ((radius + desiredTransitSize/2) *
                      Math.Sin(angle * Math.PI / 180));
15                domainPositions.Add(new Point(xPos, yPos));
16                angle += step;
17        }
18        //pro každý uzel tranzitní domény vygeneruje náhodný počet koncových domén
19        for (int transitNode=0; transitNode < rndNetwork.Nodes.Count;
              transitNode++)
20        {
21                ...
22                actStubDomains = r.Next(avgStubDomains); //náhodný počet
                      koncových domén
23                if (actStubDomains <= 0) actStubDomains = 1; //počet je alespoň 1
24
25                for (int i = 0; i < actStubDomains; i++)
26                {
27                        domPos = domainPositions[r.Next(domainPositions.Count)];
28                        ...
29                        //definice rozměrů domény
30                        lX = (domPos.X - desiredStubSize/2);
31                        hX = (domPos.X + desiredStubSize/2);
32                        lY = (domPos.Y - desiredStubSize/2);
33                        hY = (domPos.Y + desiredStubSize/2);
34                        nodes = r.Next(avgStubNodes); //náhodný počet uzlů v
                              doméně
35                        if (nodes <= 0) nodes = 1; //počet je alespoň 1
36
37                        tempNetwork = new NetworkViewModel(); //pomocný graf
38                        //vygenerování koncové domény pomocí modelu Waxman
39                        tempNetwork = waxman.RunModel(worker, nodes, (int)lX,
                              (int)hX, (int)lY, (int)hY, alfa, beta,
                              desiredStubSize);
40                        ...
41                        //vybere náhodný uzel který připojí k tranzitní doméně
42                        int stubGateNode = r.Next(tempNetwork.Nodes.Count);
43                        //spojí koncovou doménu s odpovídajícím tranzitním uzlem
44                        workNetwork.Connections.Add(new
                              ConnectionViewModel(rndNetwork.Nodes[transitNode],
                              tempNetwork.Nodes[stubGateNode],
                              ConnectionViewModel.ConnectionType.Unoriented));
45                        ...
46                        stubDomainsList.Add(tempNetwork); //seznam koncových domén
47                        ...
48                        //koncové domény jsou udržovány v pomocné síti workNetwork
49                        workNetwork.Nodes.AddRange(tempNetwork.Nodes);
50                        workNetwork.Connections.AddRange(tempNetwork.Connections);
51                }
52        }
53 }
```

Zdrojový kód 20: Vytvoření koncových domén

Na konec generování grafu ještě metoda `RunModel` přidá do grafu nadbytečné spoje. V jedné smyčce náhodně vybere ze seznamu jednu tranzitní doménu a v ní jeden uzel a jednu koncovou doménu a v ní také jeden uzel. Tyto uzly jsou následně spojeny hranou, aby simulovaly záložní spojení v síti. To samé ještě provede druhá smyčka mezi koncovými doménami. Nakonec jsou všechny pomocné grafy nakopírovány do proměnné `completeNetwork` a tato proměnná je vrácena uživatelskému rozhraní k zobrazení.

5.4.3 Charakteristiky grafu

Model Transit-Stub generuje spojité neorientované grafy jak ukazuje obrázek 29. Tabulka 3 ukazuje souhrn statistických dat o grafu z obrázku. Za zmínku stojí především relativně vysoký celkový počet uzlů n v grafu a průměr grafu vzhledem k malým hodnotám vstupních parametrů. Hustota výsledného grafu D a tedy i průměrný stupeň uzlu k jsou také nízké. Naproti tomu hustota samotných domén ať už tranzitních nebo koncových by byla vyšší vzhledem k tomu, že grafy domén jsou spojité a často i kompletní. Z obrázku je také patrná hrana mezi dvěma koncovými doménami simulující záložní spojení dvou odlišných domén.

Obrázek 29: Graf vygenerovaný algoritmem Transit-Stub.

Vstupní parametry	**n**	**e**	**Průměr grafu**	**D**	**k**
Tranzitních domén: 3 Tranzitních uzlů: 6 Koncových domén: 3 Koncových uzlů: 5	76	137	3547	0.048	3

Tabulka 3: Údaje grafu z obrázku 29

Závěr

V této práci jsem se nejdříve zabýval teorií abstraktních modelů topologií počítačových sítí a následně i jejich implementací. Čtyři modely jsem vybral tak, aby na nich bylo možné ukázat k čemu se jednotlivé typy modelů dají využít. V kapitole 3 a kapitole 4 jsem popsal výhody, nevýhody a implementaci jednotlivých modelů.

Aby bylo možné sledovat vlastnosti vygenerovaných topologií, byly modely implementovány pomocí aplikace `Nettop`, která umožňuje sledovat zároveň metriku sítě reprezentované grafem a vizuální reprezentaci spojení jednotlivých uzlů v grafu. Právě díky grafickému uživatelskému rozhraní jsou vlastnosti některé vlastnosti grafů patrné již na první pohled na rozdíl od generátorů, které poskytují uživateli třeba jen výstupní tabulku uzlů a incidentních hran.

Prostor pro zlepšení spočívá především v doplnění implementace uzlů a hran tak, aby byla umožněna například simulace chování síťových protokolů. Pro tyto případy by se mohlo jednat o doplnění vlastností jako rychlost a typ fyzické linky pro hrany nebo typy routerů a směrovací pravidla pro uzly. Dalšího zlepšení by se dalo dosáhnout zrychlením generování a zpřesněním vlastností grafů díky dodatečné optimalizaci algoritmů jednotlivých modelů.

I přes možná zlepšení, kterých by bylo možné ještě dosáhnout, se aplikaci `Nettop` daří názorně a přehledně implementovat vybrané modely topologií počítačových sítí.

Conclusions

In this thesis I focused on abstract computer network topology models' theory and their implementation. I have picked four different topology models so that it would be able to show can these different model used for evaluating different topology properties. I have described advantages and disadvantages of these models as well as their implementation in chapters 3 and 4.

All four models were implemented into an application called `Nettop` in order to be able to observe and evaluate properties of generated topologies. `Nettop` application allows the user to observe metrics of generated topologies as well as their visual representation. Unlike in some topology generators which sometimes offer only a table of nodes and connections can some of the topology properties be observed at first glance thanks to graphic user interface.

There is also room for improvement which might for example lie in extending node and edge implementation so that it can hold data needed for simulation of network protocol behavior. For example properties like the transfer speed or the type of physical links for edges and the types of routers routing policies for nodes. Another improvement can be achieved by improvement of topology generation and overall algorithm optimization for each implemented model.

Despite all possible improvements that could be made to the `Nettop` application, the application itself manages to implement and clearly and visually represent chosen computer network topology models and their properties.

A Obsah přiloženého DVD

Následuje stručný popis obsahu přiloženého DVD.

bin/
: Obsahuje adresáře `Setup` a `Portable`. Adresář `Setup` obsahuje instalátor aplikace `Nettop`. V Adresáři `Portable` se nachází aplikace Nettop ve spustitelné verzi přímo z DVD. Verze `Portable` může také vyžadovat dodatečou instalaci dalšího softwaru viz readme.txt.

doc/
: Dokumentace práce ve formátu PDF, vytvořená dle závazného stylu KI PřF pro diplomové práce, včetně všech příloh, a všechny soubory nutné pro bezproblémové vygenerování PDF souboru dokumentace (v ZIP archivu).

src/
: Kompletní zdrojové texty aplikace `Nettop` se všemi potřebnými (převzatými) zdrojovými texty, knihovnami a dalšími soubory pro bezproblémové vytvoření spustitelných verzí programu (v ZIP archivu).

readme.txt
: Soubor readme.txt obsahuje návod k instalaci a užívání aplikace `Nettop`. Stejně tak popisuje software a hardware potřebný k jejímu bezproblémovému chodu.

Navíc DVD obsahuje:

data/
: Soubory s ukázkovými topologiemi sítí vygenerovanými aplikací `Nettop`.

install/
: Obsahuje instalátory software potřebného pro bezproblémový chod aplikace. Jedná se zejména o Microsoft .NET Framework 4.5.

literature/
: Literatura využitá při vypracování bakalářské práce. Především [1] [2] [7] [8] [10] a [13].

U veškerých odjinud převzatých materiálů obsažených na DVD jejich zahrnutí dovolují podmínky pro jejich šíření nebo přiložený souhlas držitele copyrightu. Pro materiály, u kterých toto není splněno, je uveden jejich zdroj (webová adresa) v textu dokumentace práce nebo v souboru `readme.txt`.

Literatura

[1] FALOUTSOS, Michalis, Petros FALOUTSOS a Christos FALOUTSOS. *On Power-Law relationships of the Internet topology.*ACM SIGCOMM Computer Communication Review. 1999, č. 29. DOI: 10.1145/316194.316229. Dostupné z: http://dl.acm.org/citation.cfm?id=316229

[2] FALOUTSOS, Michalis, Yihua HE a Georgos SIGANOS. *Internet Topology* [.pdf]. Riverside: University fo California. Dostupné z: http://www.cs.unm.edu/ michalis/PAPERS/ITopo.pdf

[3] WÄHLISCH, Matthias, *Modeling the Network Topology* [.pdf]. FU Berlin, HAW Hamburg. Dostupné z: http://inet.cpt.haw-hamburg.de/papers/w-mnt-10.pdf

[4] JIROVSKÝ, Lukáš. *Teorie grafů* [web]. Univerzita Karlova, 2008. Dostupné z: http://teorie-grafu.cz

[5] DAJBYCH, Václav. *MVVM: Model-View-ViewModel* [web]. dotnetportal.cz, 2009. Dostupné z: http://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[6] DAVIS, Ashley. *NetworkView: A WPF custom control for visualizing and editing networks, graphs and flow-charts* [web]. codeproject.com, 2012. Dostupné z: http://www.codeproject.com/Articles/182683/NetworkView-A-WPF-custom-control-for-visualizing-a#Part1

[7] WAXMAN, Bernard M. *Routing of Multipoint Connections.* IEEE Journal on selected areas in communications [.pdf]. 1988, roč. 6, č. 9. Dostupné z: http://www.huaxiaspace.net/academic/classes/fa00/cse202/project/08Waxman.pdf

[8] CALVERT, Kenneth L., Matthew Doar a Ellen W. Zegura*Modeling Internet Topology* IPAM Workshop Tutorial [.pdf]. College of Computing, Georgia Tech, March 1997

[9] WEISSTEIN, Eric W. *Graph Diameter* From MathWorld–A Wolfram Web Resource. Dostupné z: http://mathworld.wolfram.com/GraphDiameter.html

[10] PALMER, Christopher R., Gregory J. Steffan *Generating network topologies that obey power laws* [.pdf]. Carnegie Mellon University, Dostupné z: www.eecg.toronto.edu/ steffan/papers/globecom.pdf

[11] PŘÍHODA, Petr *Počítačové sítě* [.pdf]. Přírodovědecká fakulta, Univerzita Palackého, 2007. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/poc_site.pdf

[12] ZHU Zhi-bo,GAO Fei. PLDA: *AS-level Internet Topology Generating Algorithm*. Computer Engineering, 2010, 36(7): 115-118 . Dostupné z: http://www.ecice06.com/EN/abstract/abstract9319.shtml

[13] BARABÁSI, A. - L. a ALBERT R. *Emergence of scaling in random networks.* Science. 1999, 286:509-512.