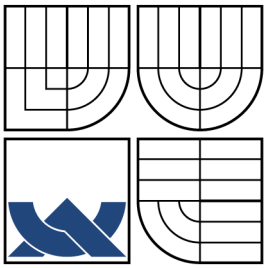
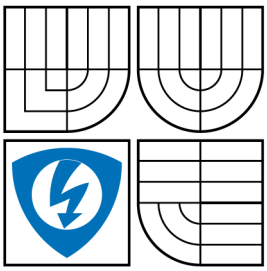


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

NALEZENÍ POZICE STANIC V INTERNETU POMOCÍ UMĚLE VYTVOŘENÝCH SOUŘADNICOVÝCH SYSTÉMŮ

INTERNET NODES LOCALIZATION USING SYNTHETIC COORDINATE SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — JAROSLAV ŠVÉDA
AUTHOR

VEDOUCÍ PRÁCE — ING. DAN KOMOSNÝ, PH.D
SUPERVISOR

BRNO 2007

ZDE VLOŽIT LIST ZADÁNÍ

Z důvodu správného číslování stránek

ABSTRAKT

Tato práce se zabývá problematikou určování zpoždění mezi dvěma síťovými uzly, např. dvěma stanicemi, dvěma servery či stanicí a serverem. Hlavním důvodem zavedení metod efektivního určování zpoždění je především eliminace zátěže sítě opakovanými přenosy dat či režií přímého měření zpoždění. Z mnoha navrhovaných metod určování zpoždění se práce zaměřuje na metody využívající umělých souřadnicových systémů s primárním zaměřením na algoritmus Vivaldi. Jsou zhodnoceny jak vlastnosti samotných metod, tak i vlastnosti jednotlivých využívaných souřadnicových systémů používaných v praxi. Zmíněna je i problematika počtu rozměrů prostoru definovaného pouze na základě dané matice zpoždění mezi uzly. Dále jsou zmíněny některé další systémy, založené na principu logického seskupení blízkých uzlů. Praktická část zahrnuje popis vyvinutého simulačního programu VivaldiMonitor, určeného pro studium chování překryvných sítí o rozsahu do několika set uzlů implementujících algoritmus Vivaldi. Součástí je zhodnocení několika simulací provedených pomocí zmíněného simulačního programu.

KLÍČOVÁ SLOVA

Umístění stanic, určování zpoždění, predikce zpoždění, predikce RTT, souřadnicové systémy, počet rozměrů, algoritmus Vivaldi, shlukování uzlů, VivaldiMonitor, simulace.

ABSTRACT

This thesis deals with predicting the latency between two network nodes, such as the two stations, two servers or server and station. The main reason for adoption of effective latency prediction techniques is the elimination of network load caused by unnecessary repeated transmissios or by direct measurement of the latency. Of the many proposed methods of latency estimation, this thesis is focused on methods using artificial coordinate systems with primary focus on the Vivaldi algorithm. Characteristics of the latency prediction methods and properties of various coordinate systems used in practice are evaluated. The issue of the number of dimensions of space defined only by the latency matrix between nodes is also mentioned. Furthermore, some other systems, based on logical clustering of nearby nodes, are mentioned. Description of simulation software VivaldiMonitor developed as part of the thesis is included. The primary purpose is analysis of the behavior of overlay networks implementing Vivaldi algorithm with less than a few hundred nodes. The Vivaldi algorithm is assessed by several simulations carried out using the aforementioned software.

KEYWORDS

Host localization, latency measurement, latency prediction, RTT prediction, coordinate systems, dimensionality, Vivaldi algorithm, node clustering, VivaldiMonitor, simulation.

ŠVÉDA, J. *Nalezení pozice stanic v Internetu pomocí uměle vytvořených souřadnicových systémů*. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2009. 85 s. Diplomová práce. Vedoucí práce Ing. Dan Komosný, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Nalezení pozice stanic v Internetu pomocí uměle vytvořených souřadnicových systémů“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Danovi Komosnému, Ph.D. za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce. Rovněž děkuji kolegovi Bc. Tomášovi Handlovi za testování a připomínky k uživatelskému rozhraní vyvinuté aplikace.

OBSAH

Úvod **11**

1 Určování polohy a vzdáleností stanic v síti Internet **13**
1.1 Význam určování polohy . . . 13
1.2 Absolutní a relativní poloha . . . 15
1.3 Zpoždění v síti a metody jejího určení . . . 16

2 Predikce zpoždění pomocí souřadnicových systémů **18**
2.1 Rozdělení používaných souřadnicových systémů . . . 19
2.2 Určení vhodného počtu rozměrů . . . 20
2.3 Systémy používající umělé souřadnicové systémy . . . 22
2.3.1 Systém Global Network Positioning (GNP) . . . 22
2.3.2 Systém Lighthouses . . . 22
2.3.3 Algoritmus Vivaldi . . . 23
2.3.4 Algoritmus Vivaldi s konstantním krokem . . . 24
2.3.5 Algoritmus Vivaldi s adaptivním časovým krokem . . . 25
2.3.6 Systém PCoord . . . 26
2.4 Strategie volby sousedů uzlu . . . 28

3 Jiné metody predikce zpoždění **30**
3.1 Systém IDMaps . . . 30
3.2 Systém Internet Iso-bar . . . 31
3.3 Metoda King . . . 32
3.4 Systém Meridian . . . 33

4 Knihovna vivaldi-lib **36**
4.1 Knihovna s pevným krokem . . . 36
4.2 Pomocné třídy . . . 38
4.3 Knihovna s adaptivním časovým krokem . . . 39

5 Vyvinutý simulační program VivaldiMonitor **40**
5.1 Popis rozhraní a ovládání . . . 41
5.1.1 Základní struktura rozhraní . . . 41
5.1.2 Načtení hotového modelu . . . 42
5.1.3 Definování a úpravy simulované topologie . . . 42
5.1.4 Spuštění a ovládání simulace . . . 44
5.1.5 Přístup k průběhům proměnných uzlů a sítě . . . 45
5.1.6 Zhodnocení predikce RTT spojů . . . 47

5.1.7 Uložení dat simulace 49
5.2 Přehled struktury aplikace 51
5.3 Charakteristika vybraných tříd 52
5.4 Děje při spuštění simulace a v jejím průběhu 54
5.5 VivaldiMonitor pro algoritmus Vivaldi s pevným časovým krokem . . 55

6 Provedené simulace **56**
6.1 Simulace uměle vytvořených sítí 56
6.2 Chování algoritmu pro reálná data 57
6.2.1 Simulace sady Meridian 57
6.2.2 Referenční simulace sady Meridian 58
6.2.3 Simulace vzorku 60 uzlů sady Meridian 61
6.2.4 Simulace vzorku 120 uzlů sady Meridian 61
6.2.5 Simulace vzorku 300 uzlů sady Meridian 62
6.2.6 Shrnutí simulací sady Meridian 65
6.2.7 Simulace sady King1 67

7 Závěr **68**

Literatura **69**

Seznam symbolů, veličin a zkratek **76**

Seznam příloh **77**

A Dokumentace simulačního programu VivaldiMonitor **78**
A.1 Spuštění simulace 78
A.2 Pozastavení a pokračování simulace 79
A.3 Diagram hlavních tříd aplikace 80
A.4 Diagram tříd modelu sítě 81
A.5 Vazby s knihovnou vivaldi-lib 82
A.6 Životní cyklus objektu Vivaldi_node2 83

B Přehledový diagram knihovny vivaldi-lib **84**

C Obsah přiloženého CD **85**

SEZNAM OBRÁZKŮ

1.1 Příklad odlišnosti logické topologie sítě od fyzické 14
2.1 Vzdálenost v 2D prostoru, 2D prostoru s výškou a 3D prostoru 21
2.2 Reprezentace predikovaného zpoždění pružinou – a) bezchybná predikce, b) nadhodnocení zpoždění o d 24
2.3 Vliv konstanty c_e na průběh místní chyby uzlu algoritmu Vivaldi e_i, poč. hodnota $e_i = 100$ 27
3.1 Zjednodušení topologie prostřednictvím tracerů systému IDMaps . . . 30
3.2 Predikce zpoždění metodou King 32
4.1 Algoritmus vlákna simulačního uzlu 37
5.1 Vzhled hlavního okna aplikace VivaldiMonitor 41
5.2 Vzhled hlavního okna aplikace VivaldiMonitor – karta *Definice uzlů* . 43
5.3 Vzhled hlavního okna aplikace VivaldiMonitor – karta *Definice spojů* 44
5.4 Aplikace VivaldiMonitor – Vzhled dialogu *Nastavení simulace* 45
5.5 Vzhled hlavního okna aplikace VivaldiMonitor během simulace 46
5.6 Vzhled hlavního okna aplikace VivaldiMonitor během simulace – stopa vývoje uzlu 47
5.7 Aplikace VivaldiMonitor – vývoj proměnných uzlu během simulace . 48
5.8 Vzhled hlavního okna aplikace VivaldiMonitor během simulace – data sítě 49
5.9 Vzhled hlavního okna aplikace VivaldiMonitor během simulace – karta *Predikce RTT* 50
6.1 Ukázka vývoje chyby sítě 225 uzlů algoritmu Vivaldi s konstantním krokem, $\delta = 0{,}3$. Převzato z [26]. 56
6.2 Vývoj chyby algoritmu Vivaldi, sada Meridian 120 uzlů, $c_c = 0{,}25$, $c_e = 0{,}05$; 0,6; 0,99 60
6.3 Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů a velikosti sítě, sada Meridian 65
6.4 Závislost relativní chyby predikce algoritmu Vivaldi na provázanosti a velikosti sítě, sada Meridian 65
6.5 Závislost chyby predikce algoritmu Vivaldi na počtu sousedů a velikosti sítě, sada Meridian 66
6.6 Závislost chyby predikce Vivaldi na provázanosti a velikosti sítě, sada Meridian 67

SEZNAM TABULEK

2.1 Srovnání vektorových operací v prostorech 2D, 2D s výškou a 3D . . 21
6.1 Závislost relativní chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů 58
6.2 Závislost absolutní chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů 59
6.3 Závislost relativní chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů s 20 sousedy 61
6.4 Závislost chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů s 20 sousedy 62
6.5 Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 60 uzlů 62
6.6 Závislost chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 60 uzlů 63
6.7 Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 120 uzlů 63
6.8 Závislost chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 120 uzlů 63
6.9 Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 300 uzlů 64
6.10 Závislost chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 300 uzlů 64
6.11 Konfigurace použité pro simulace sady Meridian 64

ÚVOD

Celosvětová síť Internet se od svých počátků bouřlivě rozvíjela a rozvoj neustále pokračuje [28]. Na tento prudký, exponenciální rozvoj počtu stanic a dalších uzlů dále navazuje stejně prudký rozvoj množství různých služeb a překryvných sítí, které se podílí na celkové zátěži sítě. Typickým příkladem jsou moderní výměnné sítě – např. BitTorrent [59], Gnutella [21], eMule [17] – vzniklé za účelem sdílení dat, poskytování různých služeb, realizaci distribuovaných výpočtů a podobně. Provoz v těchto sítích je značný a představuje podstatnou část celkového provozu v Internetu, který se od roku 1997 meziročně zdvojnásobuje [41]. Architektura používaných komunikačních modelů pak umožňuje všem uzlům jednoduše komunikovat nezávisle na skutečném propojení. Překryvné sítě pak nezřídka zrcadlí některé funkce nižších vrstev pro takto vzniklé logické spoje – nabízí často i obdobu skupinových či všesměrových datových přenosů [16], obdobu směrování [11] a podobně. Pokud komunikace uzlů nerespektuje topologii sítě, po které probíhají vlastní přenosy, je tato komunikace velmi neefektivní. To se stává problémem zejména při velkých objemech dat či multicastových přenosech. Existují sice způsoby jak aplikacím poskytnout informace o procesech na nižších vrstvách, vyžadují však podporu na straně sítě [38] a nelze je tedy použít globálně.

Multicastové přenosy jsou základem pro streamované multimediální přenosy, používané např. u internetového televizního vysílání (IPTV), přednášek a podobně. Tyto multicastové přenosy je nutné vzhledem k malé podpoře multicastových protokolů ze strany poskytovatelů realizovat na aplikační vrstvě pomocí jednotlivých přenosů bod-bod (unicast) přenosů [9]. V tomto uspořádání jednotliví spotřebitelé zároveň poskytují doručená data dalším blízkým spotřebitelům.

Aby nedocházelo k plýtvání šířky pásma, je nutné tyto příjemce vhodně volit. Jedním z prostředků je určení vzdálenosti mezi koncovými uzly, dané nejčastěji zpožděním – dobou přenosu malé datové jednotky mezi dvěma uzly. Uzly s malým zpožděním od daného uzlu jsou s největší pravděpodobností i blízko z hlediska topologie – tedy jsou propojeny přes malý počet směrovačů a dalších mezilehlých uzlů a toky mezi klienty v rámci jedné sítě tak nebudou zbytečně směrovány mimo tuto síť.

Volba na základě vzdálenosti je užitečná i v celé řadě dalších případů. V případě distribuce větších datových souborů jsou tato data vzhledem k nutnosti rozložení zátěže umístěna na několik různých serverů, tzv. zrcadel a z obdobných důvodů je třeba vybrat nejbližší zrcadlo. Jiným příkladem jsou síťové hry hrané v reálném čase, především dnes velmi populární síťové hry s velkým počtem hráčů, např. World of Warcraft [6]. Tyto přestávají být od určitého zpoždění hratelné [20] a je tedy nutné v zájmu plynulosti hraní volit nejbližší server.

K tomuto účelu bylo doposud navrženo více různých metod, jak zpoždění určit či odhadnout, zatím však neexistuje žádný standardizovaný, široce použitelný protokol. Některé metody, např. IDMaps [19] nebo Internet Iso-bar [12] dělí Internet do shluků uzlů [45] a tak jistým způsobem zrcadlí topologii Internetu. Mezi perspektivní pak lze řadit zejména metody a systémy založené na souřadnicových systémech. Mezi nejznámější patří algoritmus Vivaldi [15], Global Network Positioning (GNP) [40], Practical Internet Coordinates (PIC) [10], Internet Coordinate System (ICS). Zatím však tyto systémy zůstávají spíše na akademické půdě a na široké využití teprve čekají. Zatím nejblíže k praktickému nasazení má zřejmě služba založená na algoritmu Vivaldi, který už byl implementován do několika distribuovaných projektů [3, 50, 44]. Prakticky použitelnou alternativu k těmto systémům pak představuje např. systém King [22].

Tato práce se několika různým metodám odhadu zpoždění či obecně vzdálenosti věnuje. Podrobně je zmíněn především algoritmus Vivaldi, prostor je věnován i alternativním metodám určení zpoždění. Popisu těchto metod je věnován úvod práce, praktická část pak popisuje vyvinutý jednoduchý simulační program (aplikaci) VivaldiMonitor určený pro studium chování algoritmu Vivaldi. Součástí je i popis simulací, provedených ve spolupráci s Bc. Tomášem Handlem, který se algoritmem Vivaldi podrobně zabývá ve své práci [25, 26]. Aplikace používá jako simulační jádro knihovnu, která byla vyvinuta jako praktická část už zmíněné práce.

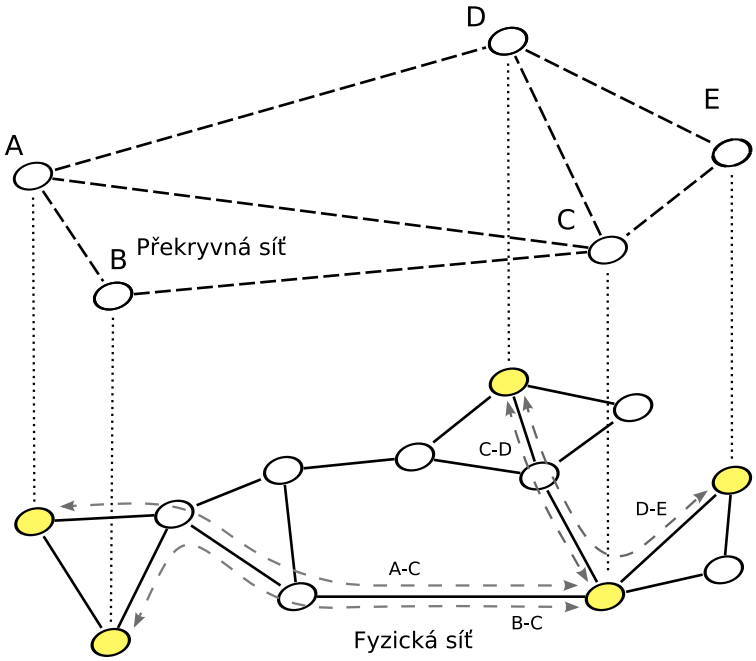
1 URČOVÁNÍ POLOHY A VZDÁLENOSTÍ STANIC V SÍTI INTERNET

1.1 Význam určování polohy

Propojení uzlů, na které je třeba data přenést, je díky dnes používaným protokolovým sadám velmi snadné. Aplikace běžící na jednom uzlu (stanici, serveru) jednoduše využije rozhraní protokolové sady, která umožní vytvořit logické spojení se vzdáleným uzlem. V prostředí nejvíce rozšířené protokolové sady TCP/IP (Transmission Control Protocol – Internet Protocol) jde o spoj na úrovni transportní vrstvy [47]. Na nejnižší vrstvě sady TCP/IP – vrstvě k přístupu k síti – tomuto přenosu po logickém spoji odpovídá několik přenosů po fyzických spojích. Pro mnohé aplikace je však velmi důležité nějakým způsobem postihnout topologii sítě umožňující jejich komunikaci. Jedná se především o služby, založené na překryvných sítích. Některé již byly zmíněny v úvodu – souhrnně jde o problematiku decentralizovaných úložišť dat, jejich distribuci (výměnné sítě) či replikaci (zrcadlení) s cílem zajistit rozložení zátěže či redundanci pro případ výpadku některého z úložišť, jako jsou například servery zpřístupňující ke stažení software, Linuxové distribuce či jakékoli jiné větší objemy dat.

Příklad rozdílu fyzické a logické topologie sítě ilustruje obrázek 1.1, zobrazující několik malých skupin uzlů, které mohou představovat např. jednotlivé sítě poskytovatelů. Některé z uzlů, propojených do topologie neúplného polygonu, spolu vzájemně komunikují a vytváří tak překryvnou síť. Každý přenos po logickém spoji je proveden přenosem po jednom (např. spoj A-B) a více spojích (např. spoj A-C). Tyto fyzické cesty mezi dvěma uzly se při nevhodné volbě uzlů, mezi nimiž jsou v rámci přenášeny data (dále označené jako *sousedé*), mohou různě překrývat. Příkladem překrývání části fyzických cest dvou odlišných logických cest jsou například spoje A-C a B-C. Logický spoj A-C je veden přes tři fyzické spoje, z nichž dva sdílí i s logickým spojem B-C. Pokud by například byl přenášen velký objem dat z uzlu A do C a posléze do B, po těchto spojích by se tato data přenesla dvakrát. Jiným příkladem jsou spoje C-D a D-E – zde přenos mezi D a E projde i přes celou trasu C-D.

Vůbec prvními překryvnými sítěmi byly tzv. distribuované hashovací tabulky (DHT) [43]. Distribuovaná hashovací tabulka je jednoduchá překryvná síť určená – stejně jako klasická hashovací tabulka – k ukládání hodnot a jejich dohledávání podle jednoznačného identifikátoru (tzv. klíče). Uzel, který je součástí DHT, je zodpovědný za uložení položek identifikovaných určitým rozsahem klíčů a je provázán s uzly, které spravují okolní rozsahy. Toto přiřazení bylo z počátku určeno jen lo-



Obr. 1.1: Příklad odlišnosti logické topologie sítě od fyzické

gickým uspořádáním DHT, bez ohledu na skutečné, fyzické propojení. Typickými příklady jsou tzv. strukturované DHT jako jsou např. Chord [54], CAN (Content Addressable Network) [49]. Primárním cílem bylo rozložení zátěže na více uzlů, efektivita byla poněkud podružná. Trpěla zejména efektivita dohledávání a směrování požadavků. Jistého zvýšení bylo dosaženo zařazením mechanizmů, které upravovaly procesy jako je určité přizpůsobení směrování na základě vyhodnocení zpoždění mezi uzly a upřednostněním uzlů (sousedů) s nízkým zpožděním. Tato úprava však má jen doplňkový charakter.

V případě velkých objemů dat – tzn. u přenosů v rámci výměnných sítí, dále distribuci multimediálních toků dat skupinám příjemců a podobně – je však třeba co nejlépe postihnout strukturu celosvětové sítě, propojující účastníky těchto sítí. Podíl tohoto typu provozu je značný, např. při měření provozu na páteřním spoji sítě France Telecom v roce 2003[1] tvořil provoz výměnných sítí asi 60 – 80 % [5]. Pokud by nebyl proveden nějaký druh optimalizace na základě zpoždění, nevyhnutelně bude docházet k opakovaným přenosům a nadměrnému zatěžování sítě, což je nežádoucí.

[1] Jednalo se o spoj s přenosovou kapacitou 1 Gb/s, zátěž byla asi 43 %.

1.2 Absolutní a relativní poloha

V předchozí části byla současně s polohou vždy zmíněna i vzdálenost. Tyto pojmy jsou v kontextu problematiky umělých souřadnicových systémů vzhledem k relativní povaze polohy velmi úzce svázány. Vzdálenost k uzlu je téměř výhradně určována na základě vzájemné polohy uzlů, která je určována pomocí určitého algoritmu z naměřených vzájemných vzdáleností. Absolutní poloha zde až na speciální případy není důležitá. Určená poloha stanice je čistě poloha logická a nemá žádnou souvislost např. se systémem zeměpisných souřadnic, který lze v běžné praxi chápat jako absolutní.

Použití systému zeměpisných souřadnic je sice možné, ale má svá omezení. Asi největším omezením je nutnost pořídit si GPS (global positioning system) přijímač. Ty jsou sice rozšířené, ale zdaleka ne dostatečně. Druhým problémem je nejednoznačná závislost zpoždění na zeměpisné vzdálenosti [23]. Zpoždění je ovlivněno několika faktory. Mezi nimi je i délka spojů mezi mezilehlými prvky – to je způsobeno konečnou rychlostí pohybu nosičů signálu, jako jsou elektrony a fotony. Délka spojů společně s konečnou rychlostí způsobí nenulovou dobu šíření signálu médiem. Ne vždy jde ale o nejdůležitější složku. Vliv doby šíření je dobře patrný u testovacích sítí jako je PlanetLab [46], jehož uzly jsou připojeny do Internetu pomocí spoje s vyšší přenosovou rychlostí. Spoje pak vykazují určitou míru korelace zpoždění a geografické vzdálenosti [29]. Z měření vyplývá, že velmi dobře korelují spoje uvnitř kontinentů, spoje mezikontinentální už ale zavádí dodatečné zpoždění, takže závislost neplatí obecně.

Při reálném nasazení jsou však součástí překryvné sítě stanice, které jsou připojeny přes přístupovou síť s nepříznivými parametry, např. malá šířka pásma nebo velký počet účastníků, jejichž provoz je koncentrován. Typickým příkladem je připojení po ADSL (asynchronous digital subscriber line), jehož zpoždění zapříčiněné čekáním paketů ve frontách může nabýt několikanásobku typické hodnoty, výjimkou není ani zpoždění nad 1000 ms [53]. Nepříznivý vliv na zpoždění v síti má také přetížení spojů. Velký provoz na spojích způsobí ukládání datových jednotek do vyrovnávacích pamětí mezilehlých prvků, což zpozdí jejich zpracování.

Rozdíl mezi oběma typy vzdáleností je ještě patrnější, není-li predikováno zpoždění. Kromě zpoždění lze použít i další veličiny, např. propustnost nebo ztrátovost spoje. Charakter těchto a dalších parametrů, sledovaných v rámci zajištění tzv. kvality služeb, není tak dobře prozkoumán. Vzdáleností se proto v dalším text rozumí zpoždění (latence). Podobně jako zpoždění však lze teoreticky použít jakýkoli parametr, splňující požadavky kladené na tzv. metriku, známou z metrických prostorů – viz dále.

1.3 Zpoždění v síti a metody jejího určení

Zpoždění mezi dvěma uzly je v užším smyslu definováno jako čas uplynulý od odeslání zprávy zdrojovým uzlem a jejím přijetím uzlem cílovým. Tato veličina je označována jako zpoždění jednosměrné. Měření jednosměrného zpoždění však není obecně použitelné, obecně vyžaduje dobrou časovou synchronizaci obou uzlů [51]. Častěji se i z tohoto důvodu používá zpoždění obousměrné, představované především často používaným parametrem RTT (Round-Trip Time). Obousměrné zpoždění zahrnuje dobu přenosu paketu od zdroje k příjemci a zpátky a také dobu zpracování paketu.

RTT k libovolnému uzlu lze jednoduše změřit přímo z lokální stanice pomocí zpráv signalizačního protokolu ICMP [29], což je standardní součást protokolové sady TCP/IP. Tuto metodu měření RTT používají například dva ze základních nástrojů správy sítě – *ping* a *traceroute* [29]. Výsledky je však třeba dále statisticky zpracovat, jelikož z principu procesu směrování v IP sítích vyplývá, že měřicí paket může být směrován zpět k odesílateli odlišnými cestami. Nejvhodnější je zřejmě medián několika měření, jelikož není ovlivněn náhodnými vzorky podstatně odlišnými od typické hodnoty [61].

Problémem určování vzdáleností na základě přímého měření je však zatížení sítě měřením výše zmíněných parametrů, které vzrůstá s počtem uzlů, které chtějí udržovat vzdálenosti k ostatním uzlům zapojeným do stejné překryvné sítě. Pro N uzlů je nutné periodicky měřit $N(N-1)$ vzdáleností, počet tedy roste kvadraticky s počtem uzlů. Toto řešení však nelze aplikovat na rozsáhlé sítě (není škálovatelné), příkladem je služba provádějící tato měření mezi 500 uzly testovací sítě PlanetLab, která byla ukončena z důvodu rychle vzrůstající režie této služby[55], která v době ukončení této služby zahrnovala asi 500 uzlů[2]. V současnosti je v provozu služba podobného charakteru, omezující provoz způsobený měřeními pomocí měření mezi reprezentanty každé lokality s uzly sítě PlanetLab.

Režii měření zpoždění mezi uzly lze značně omezit použitím systémů které vzdálenosti v síti dostatečně přesně a spolehlivě odhadují. Metod, které umožňují odhad vzdálenosti bylo představeno velké množství. Nejvíce metod využívá nepřímou predikci pomocí souřadnicových systémů. Z hlediska praktického využití lze za nejúspěšnější pokládat algoritmus Vivaldi. Jeho implementace je součástí některých DHT. Dostupná je i knihovna *pyxida* [44], umožňující velmi snadno zahrnout implementaci algoritmu do existujících projektů. Ostatní metody jsou založeny na přímých měřeních, ale s omezením kombinací komunikujících stanic. To lze zajistit sdružováním do skupin a měřením jen mezi těmito skupinami, podobně jako v uvedeném

[2] V současnosti (2009) je to asi dvojnásobek.

příkladu měření zpoždění mezi uzly sítě PlanetLab. Tento princip je i základem starších systémů jako je například IDMaps [19] a Internet Iso-bar [12]. Toto zjednodušení využívá i metoda King, která je dodnes velmi dobře použitelná.

Určitá část těchto systémů (např. IDMaps) byla navržena jako celosvětová služba s minimálními požadavky na klienty [19], podobně jako dnešní služba překladu doménových jmen (DNS – domain name service). Tyto pokusy však zůstaly pouze na akademické půdě v rámci sítě PlanetLab. Největší potenciál v tomto ohledu má zatím systém Meridian, který je zatím spuštěn na 413 uzlech PlanetLabu. Principiálně pak může sloužit jako celosvětová služba pro predikci zpoždění jakákoli překryvná síť, implementující některý ze systémů predikce pomocí souřadnicových systémů, například síť klientů Azureus. Postačilo by použít nutné součásti protokolu.

Režii měření zpoždění lze při vhodných podmínkách téměř úplně eliminovat. Klíčová je zejména pravidelná komunikace uzlů, což je například u výměnných sítí téměř vždy splněno. V takovém případě lze zpoždění velmi dobře měřit i pomocí jakékoli komunikace, při které je potvrzováno doručení zprávy, případně u komunikace typu dotaz-odpověď. V takovém případě lze výpočtem rozdílu mezi odesláním požadavku a vrácením odpovědi získat hodnotu RTT mezi uzly, podobně jako je měřeno učením prodlevy mezi vysláním zprávy *echo request* a příjmem zprávy *echo reply* protokolu ICMP.

2 PREDIKCE ZPOŽDĚNÍ POMOCÍ SOUŘADNICOVÝCH SYSTÉMŮ

Systémy založené na aplikaci umělých souřadnicových systémů přiřazují každému uzlu v síti bod v určitém metrickém prostoru. Úkolem algoritmů těchto systémů je přiřadit takové souřadnice, aby vzdálenosti mezi body představujícími uzly sítě správně predikovaly zpoždění [40]. Přiřazení je provedeno vhodným algoritmem, vycházejí nejčastěji z teorie optimalizačních algoritmů [40, 36]. Důležitá je však i volba vhodného prostoru, do něhož budou body umisťovány.

Mezi další požadavky patří škálovatelnost na libovolný počet uzlů. Se škálovatelností dále souvisí požadavek nízké režie, což vyžaduje využití stávajícího provozu aplikace v co největší míře. Využití aplikačního provozu pro měření latence nabízí dvě základní metody. První z nich je připojení hodnot přenášených mezi uzly k stávajícím aplikačním datovým jednotkám, je-li protokol dostatečně pružný a toto rozšíření umožňuje. Tohoto mechanizmu je využito např. u zpráv DHT aplikace Azureus [3], jednoho z klientů výměnných sítí. Využití stávajícího provozu odstraňuje také omezení přímých měření pomocí protokolu ICMP, které jsou z bezpečnostních důvodů často hraničními prvky sítí blokovány. Důvodem je zejména ochrana před útokem záplavou ICMP zpráv (tzv. *ICMP flood*) s cílem zahltit některý z prvků v síti.

Základním požadavkem na souřadnicové systémy použité pro odhad síťové vzdálenosti je definice samotného pojmu vzdálenosti dvou bodů d, tzv. metriky [4]. Ta je definována pro tzv. metrické prostory, které tvoří základ v technické praxi běžně používaných souřadnicových systémů, jako jsou např. kartézské, válcové či kulové souřadnice. Každá metrika podle definice musí splňovat několik podmínek:

1. **nezápornost**, tj. metrika $d(X, Y)$ je pro dva různé body X, Y kladná (pro shodné body je nulová),
2. **symetrii**, tj. metrika $d(X, Y)$ je rovna metrice $d(Y, X)$,
3. **trojúhelníkovou nerovnost**, tj. pro každou trojici bodů X, Y, Z musí platit $d(X, Y) \leq d(Y, Z) + d(X, Z)$.

Důležitá je zde zejména platnost trojúhelníkové nerovnosti. Této problematice se věnuje článek [15], uvádějící výsledky statistické analýzy počtu porušení trojúhelníkové nerovnosti. Pokud by byl podíl trojic uzlů značný, žádný ze systémů pravděpodobně nebude schopen nalézt polohy uzlů tak, aby predikce vzdáleností byla dostatečně přesná.

Jako metodika hodnocení počtu porušení sloužilo zaznamenání podílu délky nejkratší nepřímé cesty přes právě jeden mezilehlý uzel k délce přímé cesty pro každý

přímý spoj. Míra porušení byla měřena na testovacích sadách měření vycházející z měření RTT na testovací síti PlanetLab a měření metodou King. Měření metodou King má pro praktické vyhodnocení míry porušení trojúhelníkové nerovnosti větší význam, jelikož vychází z měření RTT mezi DNS servery [22]. Odráží tedy charakter Internetu lépe než PlanetLab, což je spíše akademická síť s dobrou konektivitou k páteřním sítím. Značná část dvojic uzlů z Kingovy sady trojúhelníkovou nerovnost porušuje, pravděpodobně jde ale z větší části o nepřesnost měření. Významných porušení je však relativně málo - pro porušení větší než 5 ms jde o 4,5 % všech trojic uzlů [15]. To je poměrně malý podíl a mělo by tedy být možné nalézt souřadnicový systém, který by odrážel charakter topologie celé sítě Internet.

2.1 Rozdělení používaných souřadnicových systémů

Souřadnicové systémy, teoreticky použitelné v systémech používajících umělé souřadnicové systémy lze dělit na dva základní typy:

1. **n-rozměrné eukleidovské**,
2. **neeukleidovské** – kulové, válcové, toroidní, kuželové, hyperbolické atd.

Při zkoumání schopností různých algoritmů byly zkoušeny různé typy souřadnicových systémů včetně zde uvedených, nejvíce se však používají systémy eukleidovské. Výhodou eukleidovských systémů oproti ostatním systémům – například vzhledem k tvaru Země nejpřirozenějším válcovým – je možnost jednoduše zvýšit počet rozměrů prostoru v případě, že při stávajícím počtu rozměrů není možné pro dané uzly nalézt řešení s dostatečně malou chybou. Prakticky použitelné se ukázaly 4–7D systémy [40, 52, 1]. Menší počet rozměrů obvykle neposkytuje dostatečný počet možností pro umístění uzlů tak aby měly nízkou chybu, větší počet pak zvyšuje výpočetní náročnost bez zřejmého zlepšení predikce. Není také neobvyklé, že se chyba naopak začne zvyšovat vlivem příliš velkého počtu stupňů volnosti (tzv. *curse of dimensionality*), který není možné běžně používanými metodami prozkoumat a najít vhodný stav s malou chybou predikce.

Volba správného typu souřadnicového systému je velmi důležitá, má totiž nezanedbatelný vliv na přesnost predikce. Je-li zvolen systém, který není schopen vystihnout specifika sítě, a nenabízí tak dostatek možností pro rozmístění uzlů, i dobře navržený a odzkoušený algoritmus nebude schopen odvodit souřadnice bodů tak, aby byly predikce dostatečně přesné.

Velmi zajímavý je výsledek opakovaně provedených analýz hlavních složek (PCA – principal component analysis), který odhalil pouze dvě výrazné složky [13, 1], což by znamenalo možnost umístění všech uzlů do roviny. Tento souřadný systém byl

schopný zpoždění predikovat uspokojivě, predikce při použití většího počtu rozměrů přinášejí určité, ale v daném případě nepříliš výrazné zlepšení. V praxi se však při experimentování s klienty Azureus ukazuje, že tyto rozměry navíc mohou predikci výrazně zlepšit. Po zvýšení počtu rozměrů na 4 poklesla relativní chyba o 43 % [35].

Kulové souřadnice, ač bližší zeměpisným souřadnicím zmiňovaným v úvodu, poskytovaly dostatečnou přesnost pouze po velké poloměry a se zvětšujícím poloměrem se odhad stále zpřesňoval [15]. To odpovídá přibližování se umístění na ploše, umělý model bodů rozmístěných a přímo propojených by měl malou chybu pouze pro malý interval poloměrů. To je důsledkem koncentrace uzlů převážně do Evropy, Severní Ameriky a Japonska, které mají také velmi dobrou vzájemnou konektivitu [58, 1]. Je však možné, že s dalším budováním interkontinentálních spojů – „omotáním" kolem zemského povrchu – se topologie podstatně změní a pak už tento model přestane vyhovovat. Vzhledem k neobydlenosti polárních oblastí jsou ale logičtější spíše válcové souřadnice.

2.2 Určení vhodného počtu rozměrů

Doplněním k převážně empirickému přístupu k hledání vhodného systému je pak matematická analýza charakteru prostoru, vytvořeného vzdálenostmi mezi uzly. Výhodou je oproštění se od nutnosti použít při analýze konkrétní souřadnicový systém o daném počtu rozměrů. Určení počtu rozměrů se v tomto případě stanoví na základě vývoje počtu uzlů vzdálených nejvýše r od zvoleného uzlu. Princip je následující: pokud je zkoumaná sada vzdáleností dvourozměrná, tedy lze-li body umístit do roviny, lze předpokládat, že počet bodů p vzdálených nejvýše r bude úměrný ploše takto vytvořeného kruhu, tedy poroste s druhou mocninou vzdálenosti (poloměru) r. Podobně pro body umístitelné do 3D prostoru počet poroste úměrně r^3, pro prostor s d rozměry pak obecně r^d. Počet rozměrů lze díky této závislosti určit jako směrnici závislosti $\log p$ na r [1].

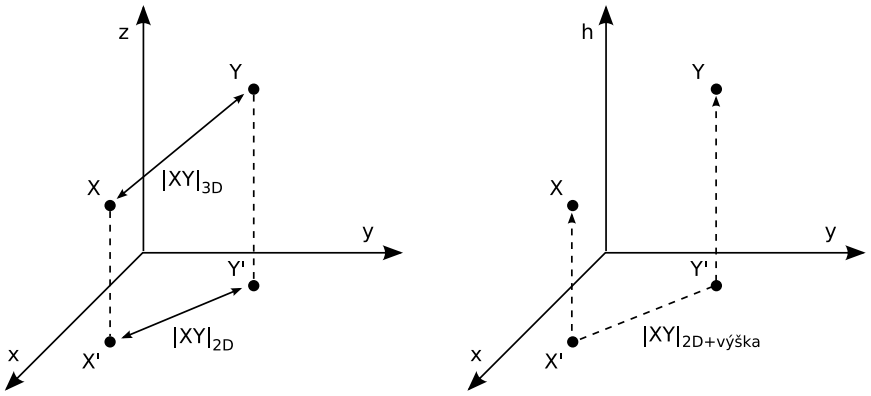
Podrobením dostupných sad měření této analýze ukazuje, že podstatná část měření zpoždění (3 – 100 ms) odpovídá 2D prostoru. To je v souladu s analýzou hlavních složek. Spoje s malým zpožděním pak tvoří část závislosti s větší směrnicí, tj. pro postihnutí této části spojů je zřejmě třeba výše zmíněné dodatečné rozměry.

Důležitým faktorem ovlivňujícím přesnost predikce je též schopnost postihnout zpoždění v přístupové síti. Připojení části stanic – například počítačů v domácnostech – vnáší často značné zpoždění. Datová jednotka odeslaná jinému uzlu musí projít páteřní i oběma přístupovými sítěmi. Jako efektivní a elegantní řešení se ukázalo přidání zvláštní složky označované jako výškový vektor (výška) [15]. Jde o další stupeň volnosti stejného charakteru jako je zpoždění přístupové sítě. Pro ilustraci

Tab. 2.1: Srovnání vektorových operací v prostorech 2D, 2D s výškou a 3D

Operace	2D	2D + výška	3D
$\mathbf{x} - \mathbf{y}$	$(x_1 - y_1, x_2 - y_2)$	$(x_1 - y_1, x_1 - y_2, x_h + y_h)$	$(x_1 - y_1, x_2 - y_2, x_3 - y_3)$
$\|\mathbf{x}\|$	$\sqrt{x_1^2 + x_2^2}$	$sqrtx_1^2 + x_2^2 + x_h$	$\sqrt{x_1^2 + x_2^2 + x_3^2}$
$a\mathbf{x}$	(ax_1, ax_2)	(ax_1, ax_2, ax_h)	(ax_1, ax_2, ax_3)

charakteru této složky je v tab. 2.1 uveden rozdíl mezi 2D prostorem, 2D prostorem s výškou a 3D prostorem formou definic základních operací s vektory v každém z prostorů. Souřadnice bodů jsou v dalším textu vždy chápány vektorově, tedy jako vektor **c** se složkami rovnými jednotlivým souřadnicím bodu. Geometricky je pojem vzdálenosti pro tyto prostory znázorněn na obrázku 2.1.



Obr. 2.1: Vzdálenost v 2D prostoru, 2D prostoru s výškou a 3D prostoru

Vliv tohoto výškového vektoru je patrný na již zmíněném výzkumu chování algoritmu Vivaldi, implementovaného v klientovi Azureus. Cílem toho výzkumu bylo především sledování vybraných ukazatelů kvality predikce. Použití 5D prostoru nepřineslo žádné zlepšení oproti tehdy použitému 2D prostoru s výškovým vektorem. Naopak pouze 4D prostor s výškovým vektorem umožnil podstatné zlepšení predikce jak z hlediska chyby predikce, tak z hlediska stability [35].

2.3 Systémy používající umělé souřadnicové systémy

2.3.1 Systém Global Network Positioning (GNP)

Systém GNP využívá systému souřadnic, vytvořeného množinou N zvláštních, tzv. orientačních bodů (OB, angl. *landmarks*) $O = O_1, O_2, \ldots O_N$ [40]. Orientační body měří zpoždění ke všem ostatním OB – což je pro řádově několik desítek těchto proveditelné bez velké zátěže sítě – a vytvoří tak matici vzdáleností $\mathbf{L}$, jejíž prvky L_{ij} odpovídají zpoždění mezi O_i a O_j, kde $i, j \in 1, 2, \ldots N$. Následně je každému O_i přiřazen bod $\mathbf{c}_i$ v eukleidovském prostoru. Přiřazení je provedeno jako výsledek minimalizace chyby

$$E_1 = \sum_i \sum_j \mathcal{E}(L_{ij}, |\mathbf{c}_i \mathbf{c}_j|), \tag{2.1}$$

kde $\mathcal{E}$ je některá z chybových funkcí, jako např. kvadratická odchylka. Jde o typickou optimalizační úlohu, při které lze použít různých algoritmů k nalezení extrému funkce. V původní práci [40] je použit jeden ze simplexních algoritmů – konkrétně Nelder-Meadův, v zahraniční literatuře známý jako Simplex Downhill [33]. Po přiřazení souřadnic je služba připravena k provozu.

Klient této služby K_1 postupuje při určování svých souřadnic podobným způsobem - změří vzdálenosti k několika OB, získá jejich souřadnice a na základě optimalizace určí vlastní souřadnice $\mathbf{c}_{k1}$. Po určení souřadnic pak stačí k odhadu vzdálenosti ke vzdálenému klientovi se souřadnicemi $\mathbf{c}_{k2}$ určit vzdálenost $\|\mathbf{c}_{k1}\mathbf{c}_{k2}\|$.

2.3.2 Systém Lighthouses

Systém Lighthouses sdílí s GNP koncept OB, které jsou zde označené jako majáky (lighthouses). Princip určení souřadnic je však principiálně odlišný a vyžaduje po změření všech vzdáleností pouze jednoduché operace lineární algebry. Podstatnou vlastností je také architektura blízká typickým překryvným sítím, každý uzel vyžaduje ke svému umístění v souřadné soustavě malou část uzlů, což umožňuje neomezenou škálovatelnost. Každý klient může na základě vzdáleností a souřadnic několika OB učit své souřadnice a dále pak sloužit jako OB pro nově se zapojující uzly. Každý uzel také udržuje adresy dalších OB, např. těch, které použil pro určení svých souřadnic. To umožní libovolnému novému uzlu se zapojit při znalosti jakéhokoli z uzlů. Více o této službě viz [42].

2.3.3 Algoritmus Vivaldi

Algoritmus Vivaldi využívá pro stanovení vhodných souřadnic minimalizaci chybové funkce shodně jako GNP. Podstatným rozdílem je však existence jednoho typu uzlů, které na sebe navzájem působí a snaží se umístit tak, aby celá síť vykazovala minimální dosažitelnou chybu (jde o metodu tzv. *relaxace pružin*). To dosahuje simulací systému hmotných bodů $U = \{U_i\}$, propojených pružinami. Propojení pružinami odráží topologii sítě uzlů, které využívají algoritmu pro predikci zpoždění. Analogie sítě pružin přiřazuje každému spoji pružinu, jejíž klidová délka je rovná změřené vzdálenosti mezi uzly [13].

Potenciální energie pružiny je úměrná druhé mocnině natažení, proto je jako chybová funkce, kterou se algoritmus snaží minimalizovat, zvolen součet kvadratických odchylek

$$E = \sum_j \sum_i (L_{ij} - \|\mathbf{c}_i - \mathbf{c}_j\|)^2, \tag{2.2}$$

kde $\mathbf{c}_i$, $\mathbf{c}_j$ jsou souřadnice bodů přiřazených uzlům U_i, U_j a L_{ij} je zpoždění mezi nimi naměřené.

Na základě této analogie sítě pružin je zvolen i algoritmus minimalizace, který vychází z Hookeova zákona. Hookeův zákon $\mathbf{F} = -k\mathbf{d}$ [24] klade přímou úměru mezi prodloužení pružiny $\mathbf{d}$ a vratnou silou $\mathbf{F}$, snažící se vrátit pružinu do klidového, nenapjatého stavu, viz obrázek 2.2. Vektor $-\mathbf{d}$ je v tomto případě třeba vytvořit z dvou částečných informací – první z nich velikost a souhlasnost orientace se směrem natažení pružiny, daná výrazem $L_{ij} - \|\mathbf{c}_i - \mathbf{c}_j\|$, druhou je informace o směru natažení pružiny, který je určen jednotkovým vektorem

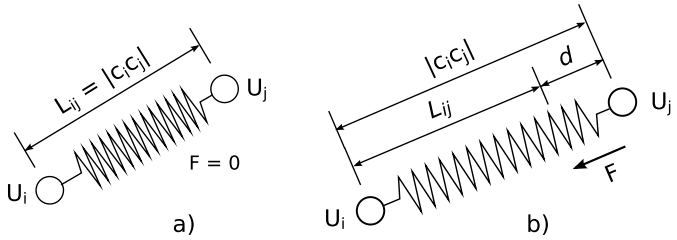
$$\mathbf{u} = \frac{\mathbf{c}_i - \mathbf{c}_j}{\|\mathbf{c}_i - \mathbf{c}_j\|}, \tag{2.3}$$

zapisovaného dále jako $\mathrm{u}(\mathbf{c}_i - \mathbf{c}_j)$. Síla působící na bod U_i jako důsledek natažení pomyslné pružiny mezi U_i a U_j je tedy rovna

$$\mathbf{F}_{ij} = (\|\mathbf{c}_i - \mathbf{c}_j\| - L_{ij}). \tag{2.4}$$

Jde o poněkud nezvyklý zápis, ale platí obecně pro libovolnou trojici $(\mathbf{c}_i, \mathbf{c}_j, L_ij)$. Z uvedeného zápisu je rovněž zřejmá hodnota konstanty úměrnosti $k = 1$. Fyzikálně jde u tuhost pružiny.

Vzhledem k distribuovanému charakteru algoritmu Vivaldi je použit iterativní algoritmus minimalizace. Za předpokladu znalosti všech souřadnic uzlů $\mathbf{C} = \mathbf{c}_1, \mathbf{c}_2, \ldots \mathbf{c}_n$ a matice vzdáleností $\mathbf{L}$ lze použít následující centralizovaný algoritmus, jehož decentralizovaná podoba pak určuje podobu algoritmu Vivaldi.



Obr. 2.2: Reprezentace predikovaného zpoždění pružinou – a) bezchybná predikce, b) nadhodnocení zpoždění o d

Tento centralizovaný algoritmus u každého uzlu vypočte celkovou sílu $\mathbf{F}_i$ na něj působící, která je dána jejich vektorovým součtem

$$\mathbf{F}_i = \sum_{j \neq i} \mathbf{F}_{ij}. \tag{2.5}$$

Tato síla působí v každé iteraci po malý čas Δt, což způsobí posun bodu o $\Delta t \mathbf{F}_i$ na nové souřadnice $\mathbf{c}_i = \mathbf{c}_i + \Delta t \mathbf{F}_i$. Popsaný postup je pak opakován pro všechny uzly, dokud chyba E neklesne pod určitou hodnotu.

2.3.4 Algoritmus Vivaldi s konstantním krokem

Pokud upravíme algoritmus na reálné podmínky, tedy znalost jen několika málo uzlů a jejich vzdálenosti, získáme nejjednodušší algoritmus Vivaldi. Každý uzel pak samostatně simuluje svůj pohyb v prostoru na základě příchozích „vzorků“ zpoždění L_j a souřadnic známých uzlů $\mathbf{c}_j$. Index i není dále používán, jelikož výpočet je svázán s uzlem U_i, na kterém probíhá. Je tak zdůrazněna distribuovanost algoritmu.

Tyto informace však nemusí obecně dostávat pravidelně a stejně často, zejména pokud algoritmus dostává informace od některé z aplikací. Při příchodu páru $(L_j, \mathbf{c}_j)$ proběhne jedna iterace algoritmu, který lze zapsat pomocí pseudokódu takto:

```
inicializace:
c = 0

vivaldi_simple(L_j, c_j):
    urči velikost chyby predikce:
    ch = ||c - c_j|| - L_j
    urči směr působení síly v důsledku chyby:
    s = u(c - c_j)
```

výpočet síly působící na hmotný bod:
$\mathbf{F} = ch \cdot \mathbf{s}$
posuv o malý krok úměrný síle:
$\mathbf{c} = \mathbf{c} + \delta\mathbf{F}$

Z tohoto zápisu je zřejmé, že jde vlastně o jednu iteraci vnitřní smyčky centralizovaného algoritmu. Jedinou změnou je odstranění indexů i a přeznačení Δt na δ. I v tomto případě je vypočtena síla působící na bod a jeho souřadnice je posunuta úměrně této síle ve směru jejího působení. Konstanta δ je označována jako časový krok (ČK), odtud označení tohoto algoritmu.

Standardně jsou uzly na počátku umístěny v počátku souřadnic. To však znamená, že některé body by mohly uvíznout, pokud by všechny jejich známé uzly měly též nulové souřadnice. Z tohoto důvodu se směr pro nulový vektor u($\mathbf{0}$) definuje jako náhodný jednotkový vektor [15], takže výsledkem je odskok obou uzlů na náhodný bod jednotkové kružnice se středem v počátku. V praxi je ale takový stav nepravděpodobný, protože jednotlivé instance algoritmu budou spuštěny v různých časových okamžicích a vždy se budou zapojovat už do existující sítě.

2.3.5 Algoritmus Vivaldi s adaptivním časovým krokem

Centralizovaný algoritmus relaxace pružin a z něho odvozený jednoduchý algoritmus Vivaldi uvedený výše není příliš praktický pro reálné nasazení. Bez dalších opatření má při simulaci reálných podmínek v síti tendenci oscilovat vlivem výskytu neustálených uzlů. Další nepříjemnou vlastností je pak schopnost každého nově se připojivšího uzlu narušit stávající síť [15], což by umožnilo velmi snadno takovou síť sabotovat. Je tedy nutné upravit algoritmus tak, aby velmi rychle nalezl své hrubé umístění a poté krok postupně snižoval. Zmíněné problémy jsou u algoritmu Vivaldi řešeny zavedením jednoduchého váhování základního časového kroku. Aktualizace souřadnic uzlu zůstává shodná, kód řádku 4 shrnuje celé tělo algoritmu Vivaldi s konstantním krokem, který tento krok rozepisoval na dílčí kroky. Doplněný algoritmus lze symbolicky zapsat následovně:

inicializace:
$\mathbf{c} = \mathbf{0}$

vivaldi(L_j, $\mathbf{c}_j$, e_j):
1 – urči váhu vzorku:

$w = e_i/(e_i + e_j)$
urči relativní chybu predikce zpoždění vzhledem k novému vzorku
$e_s = |\|\mathbf{c} - \mathbf{c}_j\| - L_j|/L_j$
2 – akualizuj hodnotu e_i
$e_i = e_s c_e w + e_i(1 - c_e w)$
3 – nastav časový krok
$\delta = c_c w$
4 – krok algoritmu s konstatním krokem
$\mathbf{c} = \mathbf{c} + \delta(L_j - \|\mathbf{c} - \mathbf{c}_j\|)\mathrm{u}(\mathbf{c}_i - \mathbf{c}_j)$

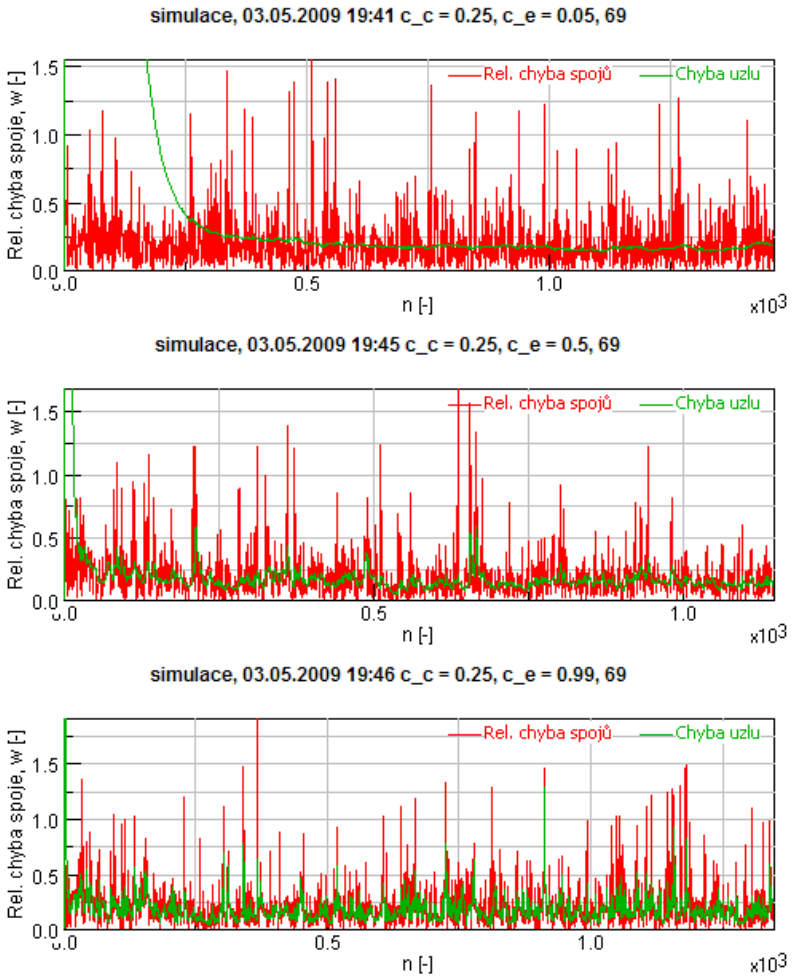
Tento algoritmus váhuje základní časový krok c_c váhou w (viz krok 3), která je ovlivněna jak chybou e_i místního uzlu, tak chybou e_j uzlu vzdáleného. Je-li chyba vzdáleného uzlu velká ve srovnání s místní chybou, jmenovatel je velký, takže výsledkem je malá hodnota váhy – daný soused je nedůvěryhodný a uzel se posune jen málo. V případě souseda s nízkou chybou je člen e_j zanedbatelný a váha je blízká jedné, takže uzel se poddá tlaku myšlené pružiny. Horní mez kroku pak určuje parametr c_c. Váha w tak vlastně představuje tuhost pružiny z Hookeova zákona, která je pro uzel tím menší, čím lépe predikuje zpoždění relativně k druhému uzlu.

Chyba uzlu e_i je určována v kroku 2 jako plovoucí vážený průměr relativní chyby spoje, označené jako e_s. Vlastnosti tohoto plovoucího průměru lze měnit pomocí ladící konstanty c_e. Je-li blízký jedné, chyba je určena okamžitou hodnotou e_s, snižováním hodnoty c_e získávají váhu minuté vzorky a vzorky jsou integrovány. Ukázku průběhu e_s a e_i pro hodnotu blízkou jedné a blizkou nule zachycuje obrázek 2.3. Červený průběh představuje e_s, zelený pak e_i. Grafy pochází ze vyvinuté simulační aplikace VivaldiMonitor, která je popsána v kapitole 5.

2.3.6 Systém PCoord

Koncept úpravy velikosti změny souřadnic podle váhy určené relativní chybovostí každého jeho souseda dále rozvíjí práce [36]. Popsaný algoritmus PCoord používá při každé aktualizaci konstantní počet vzorků, což umožňuje lépe zhodnotit míru spolehlivosti predikce uzlů a tedy i koeficient časového kroku.

Princip algoritmu PCoord je následující. Při každé iteraci algoritmu v uzlu U_i je zvoleno M sousedů uzlu z množiny všech známých sousedů. Těchto M uzlů je pak zpracováváno během jedné iterace jako celek, dávka. Od každého z uzlů dávky (označeného jako U_j) jsou následně zjištěny současné hodnoty souřadnic $\mathbf{c}_j$, zpoždění L_j, plovoucího průměru chyby predikce $e_{\mathrm{p},j} \in \langle 0; 1\rangle$ a plovoucího průměru zbytkové chyby optimalizace (termín bude vysvětlen dále v této části). Pro zvýšení stability



Obr. 2.3: Vliv konstanty c_e na průběh místní chyby uzlu algoritmu Vivaldi e_i, poč. hodnota $e_i = 100$

algoritmu je do množiny zahrnut i záznam uzlu U_i, vzdálenost L_i je pochopitelně nulová. Uzel U_i tak klade odpor úměrný své váze, takže bod dobře predikující vzdálenosti k ostatním není zmítán uzly s velkou chybou, kterým jsou naopak přiřazovány váhy malé. Na základě chyb uzlů dávky jsou dále určeny jejich váhy

$$w_j = \frac{a_j^2}{\sum_{k=1}^{M} a_k^2}, \tag{2.6}$$

kde koeficienty $a_k = 1 - -e_{\mathrm{p},j} + \tau$, τ je konstanta blízká nule přiřazující velmi malou (avšak nenulovou) váhu uzlům s velkou chybou. Výhodou tohoto váhování je velký důraz na uzly s nízkou chybou – takové mají velkou váhu a potlačují roli ostatních. To je dáno konečnou hodnotou součtu vah pro libovolnou velikost dávky (je zřejmé, že součet $\sum w_j = 1$) a dále kvadratickými členy, které malé hodnoty a_n potlačují podstatně více než hodnoty velké.

Vlastní výpočet nových souřadnic $\mathbf{c}_\mathrm{n}$ je pak proveden některou z optimalizačních metod pro chybovou funkci

$$\mathcal{E} = w_i(L_i - \|\mathbf{c}_\mathrm{n} - \mathbf{c}\|)^2 + \sum_{j=1}^{M} w_j(L_j - \|\mathbf{c}_\mathrm{n} - \mathbf{c}_j\|)^2. \tag{2.7}$$

Počet kroků optimalizačního algoritmu ani samotný optimalizační algoritmus udán striktně není, tj. PCoord umožňuje velmi jednoduše využít jiný optimalizační algoritmus a jeho nastavení. Provedení více než jedné iterace optimalizačního algoritmu, podobně jako u systému GNP[40], je asi největší rozdíl mezi algoritmy Vivaldi a PCoord. Tato volba pak umožňuje dosáhnout rychlejší konvergence – předstihuje i algoritmus Vivaldi.

Vzhledem k tomu, že optimalizační algoritmy často naleznou pouze lokální extrém (v tomto případě minimum), chybová funkce pro dané $\mathbf{c}_j$ nebude nulová. Tato hodnota – zbytková chyba optimalizace e_f – určuje, do jaké míry jsou údaje dávky, s níž optimalizace pracovala, konzistentní. Pokud jsou údaje protichůdné – např. mezi některými uzly je zásadně porušena trojúhelníková nerovnost – optimalizace skončí s vysokou chybou. V takovém případě je zřejmé, že není moudré klást takovým souřadnicím velkou váhu. Pokud ovšem je výsledná chyba malá, pravděpodobně se jedná o změnu která pomůže k rychlé minimalizaci uzlu a potažmo sítě jako celku.

Jako ukazatel konzistence slouží poměr

$$\rho = \min\left(\frac{e_\mathrm{f}}{e_\mathrm{f,n}}, 1\right), \rho \in \langle 0; 1\rangle \tag{2.8}$$

kde e_f je zbytková chyba z minulé iterace, $e_\mathrm{f,n}$ pak zbytková chyba iterace současné. Role této chyby má podobnou funkci jako ČK algoritmu Vivaldi – $\mathbf{c}_\mathrm{n}$ zjištěné optimalizací určuje pouze směr, kterým se má souřadnice posunout, velikost tohoto posuvu je dána koeficientem ρ. Popsaný algoritmus aktualizace v podstatě provádí „průzkum“ krajiny chybové funkce kolem aktuálních souřadnic uzlu za účelem rozhodnutí, zda se vyplatí souřadnici změnit.

2.4 Strategie volby sousedů uzlu

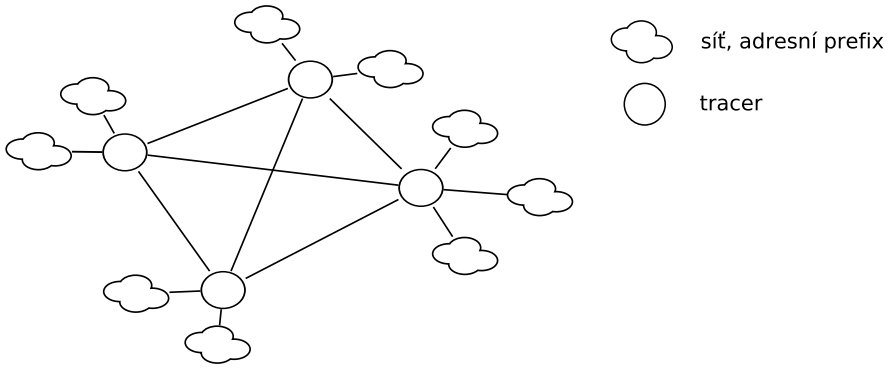
Kromě volby souřadnicového systému a algoritmu predikce záleží u simulačních algoritmů také na strategii volby sousedů, které uzel použije pro stanovení polohy. Tyto strategie jsou v principu čtyři – výběr jen blízkých uzlů, náhodný výběr, výběr části (např. poloviny) blízkých uzlů s náhodným výběrem zbývajícího počtu [36] a volba jen vzdálených. Všechny tyto strategie předpokládají dostatečně velkou množinu sousedů, z které jsou uzly voleny.

Z těchto se ukazuje být nejvhodnějším možným výběrem volba části uzlů které jsou blízko a části náhodně, tato strategie dosahuje všeobecně nejnižší výslednou predikční chybu a rychlou konvergenci . Uspokojivé výsledky má i náhodný výběr, poněkud horší ale použitelný je i výběr pouze z blízkých sousedů [36]. Pokud jsou v množině pouze uzly v nejbližším okolí daného uzlu, je pravděpodobné, že ačkoli dojde k dobrému odhadu zpoždění k těmto uzlům, globální vzdálenosti mohou být značně zkresleny [15] a tedy nebudou jako odhad použitelné. Volba jen vzdálených sousedů je také nevhodná, protože tyto uzly jsou s velkou pravděpodobností zatíženy velkou chybou, oscilují a tedy podstatně zpomalují konvergenci.

3 JINÉ METODY PREDIKCE ZPOŽDĚNÍ

3.1 Systém IDMaps

Systém IDMaps (*internet distance maps service*) byl jedním z prvních systémů, který se snažil řešit známé problémy spojené s aktivním měřením vzdálenosti v rámci každého uzlu sítě. Aby všechny uzly zapojené do Internetu měly možnost určit vzdálenost k dalším uzlům bez tohoto nárůstu, bylo navrženo měření delegovat na podstatně menší počet zvláštních, k tomu určených uzlů. Služba IDMaps je označuje jako *tracery*. Vzdálenost mezi dvěma stanicemi je pak dána součtem vzdálenosti dotazující se stanice k nejbližšímu traceru, vzdálenosti stanice k níž je vzdálenost odhadována k nejbližšímu traceru a vzdálenosti mezi oběma tracery. Tyto vzdálenosti (latence) jsou dále distribuovány tzv. HOPS (host proximity service) serverům, které pak poskytují vlastní službu predikce vzdálenosti koncovým stanicím.



Obr. 3.1: Zjednodušení topologie prostřednictvím tracerů systému IDMaps

Značného snížení počtu měření můžeme dosáhnout umístěním tracerů ke každému adresnímu prefixu (AP)[1] nebo autonomnímu systému (AS)[2]. Obě řešení mají stále své problémy – AP je sice řádově méně než všech uzlů, přesto je jich stále mnoho, zatímco AS je dostatečně malý počet, ale liší se značně svým rozsahem – některé mají globální charakter – a rozložením – nezřídka několik AS pokrývá podobou geografickou oblast. Tracery systému IDMaps jsou proto spojeny každý s několika AP a také navzájem pomocí tzv. virtuálních spojů, jak je naznačeno na obrázku 3.1. Tracery pak musí být umístěny v souladu s topologií Internetu tak, aby

[1] Adresní prefix je množina uzlů, které mají vzájemné zpoždění tak malé, že se uzlům mimo uzly AP jeví stejně vzdálené, tedy např. malá lokální počítačová síť.

[2] AS je síť nebo soustava propojených sítí, spravovaná jednou organizací (např. poskytovatelem připojení) a společnou směrovací politikou.

byla dosažena dostatečná přesnost odhadu. Jde o poměrně komplikovanou optimalizační úlohu řešící úlohy k-HST (*k hierarchically well separated trees*) a *minimum K-center* známé z teorie grafů. Řešení těchto úloh má více aplikací, mezi které patří např. určení vhodného rozmístění hasičských stanic.

Práce [19] uvádí, že pokud jsou tracery rozmístěny správně, stačí 0,2% podíl tracerů vzhledem k celkovému počtu uzlů Internetu k získání dostatečně přesných odhadů. Pro dnešní počet řádově milionů uzlů se tedy jedná o řádově tisíce tracerů. Podle měření v [22] však IDMaps má tendenci poměrně výrazně nadhodnocovat vzdálenosti, přičemž podíl nadhodnocení neklesá s velikostí nadhodnocení.

Ke zmíněnému problému přispívá zřejmě problém rozšiřování sítě tracerů, které musí být umístěny mimo jednotlivé AP, tedy zřejmě v sítích poskytovatelů a vyžadují tedy jejich spolupráci. Tato služba je navíc prakticky mrtvá. Zřejmě jediný přístup k adresám serverů služby poskytoval server closestserver.com, v současnosti se na něm původní stránky nenalézají. Největším problémem je však zřejmě potřeba změn v sítích poskytovatelů. Tento problém je řešen např. dále popisovanou metodou King, která využívá už existující infrastruktury služby DNS.

3.2 Systém Internet Iso-bar

Mnoho z výše zmíněných problémů služby IDMaps elegantně řeší systém Internet Iso-bar, využívající k predikci techniky shlukování koncových stanic, které jsou součástí systému. Koncové uzly jsou seskupovány na základě podobnosti jejich umístění vůči množině orientačních bodů[3]. Následně jsou na základě těchto údajů vytvořeny shluky, u nichž je kladen důraz na minimalizaci jejich průměru.

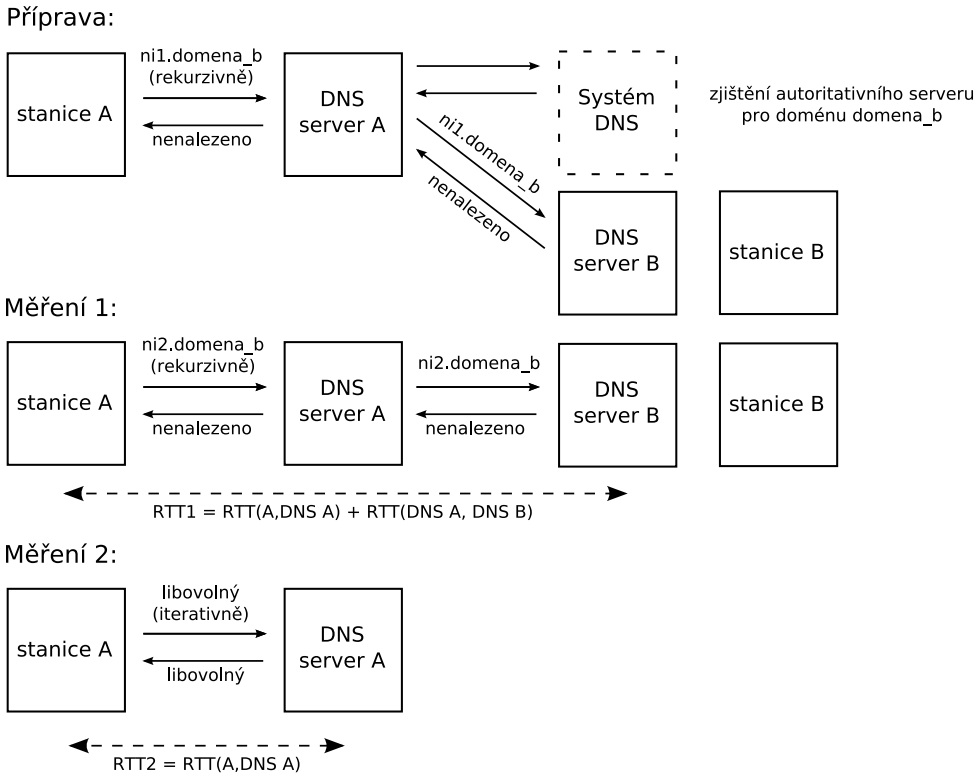
Z koncových uzlů v jednotlivých shlucích je následně vybrán jeden jako tzv. monitor, který zastává funkci velmi podobnou traceru u IDMaps. Oproti traceru navíc periodicky aktivně měří zpoždění ke koncovým uzlům a ostatním monitorům a tato měření průměruje. Podobně jako tracery jsou monitory zodpovědné za predikci zpoždění. Pro dva koncové uzly I, J v různých shlucích je jejich vzájemné zpoždění predikováno pomocí zpoždění mezi příslušnými monitory M_i, M_j těchto shluků. Pokud se oba koncové uzly nacházejí ve stejném shluku, je vzájemné zpoždění mezi I, J predikováno podle vztahu $L_{ij} = \frac{L_{im}+L_{jm}}{2}$, kde M je monitor jejich shluku. Jde tedy o méně pesimistický odhad než prostý součet zpoždění ke společnému traceru v případě systému IDMaps.

Výhodou tohoto přístupu je řešení predikce nevyžadující změny u poskytovatele a podstatně přesnější odhady než IDMaps, ve srovnání se současnými technikami je

[3]Role OB je zde odlišná od role OB u systémů jako je GNP, nepodílejí se na predikci zpoždění přímo.

tato přesnost poněkud horší. Výhodou je okamžitá odpověď monitoru, který minulá měření průměruje a odpovídá vždy hodnotou tohoto průměru.

3.3 Metoda King



Obr. 3.2: Predikce zpoždění metodou King

Velmi zajímavý a do značné míry univerzální a přesný systém odhadu RTT představuje nástroj nazvaný King [22]. Tento nástroj je založen na vynalézavém způsobu využití už existující služby DNS, která provádí překlady alfanumerických logických adres, tzv. doménových jmen (např. www.google.cz) na IP adresy síťových uzlů [34], které mají být pod daným jménem zpřístupněny.

Metodika predikce RTT je založena na předpokladu přítomnosti alespoň jednoho DNS serveru blízko (tj. majícího malou latenci) každé stanici v Internetu. Platí-li tento předpoklad[4], latenci mezi dvěma stanicemi A, B lze velmi přesně[5] odhadovat měřením latence mezi jim nejbližšími DNS servery DNS A, DNS B – viz obrázek

[4]Měřením bylo zjištěno, že 80 % latencí od klientů výměnné sítě Napster k nejbližímu DNS serveru bylo menších než 10 ms (2002) [22].

[5]75 % měřených spojů vykazovalo relativní chybu než 20 % [22].

3.2. Tuto latenci pochopitelně nelze změřit přímo. Měření se tedy provádí pomocí běžných DNS dotazů.

Pro účel vysvětlení předpokládejme, že stanice A potřebuje odhadnout vzdálenost ke stanici B. Ta provede dvě měření: první je provedeno rekurzivním DNS dotazem na server DNS B na libovolné doménové jméno v doméně serveru DNS B. Toto jméno má být nejlépe dlouhý numerický či alfanumerický řetězec, označený v obrázku 3.2 jako `ni2`. Cílem takto sestaveného dotazu je donucení DNS serveru k dotazu na dané doménové jméno, jelikož je nepravděpodobné, že by tento dotaz už dříve odpovídal a měl jeho výsledek uložený v paměti. Server DNS A tak nezná IP adresu této (fiktivní) stanice, požadavek je tedy přeposlán serveru B. V praxi po příjmu požadavku obvykle nezná adresu serveru překládajícího doménová jména pro doménu, v níž se druhá stanice nachází. Jeho adresu lze zjistit pomocí opakovaných dotazů – viz horní část obrázku 3.2. Tento proces vyhledávání způsobí odklad odeslání odpovědi na požadavek a zkreslení odhadu zpoždění, proto je vykonán v rámci přípravy k vlastnímu měření. Tento dotaz je třeba vést na jiné doménové jméno (v tomto případě jiný náhodný řetězec `ni2`) než jméno v dotazu použitém k měření RTT k serveru B.

Server B odpoví na tento dotaz – odpověď bude až na výjimky pochopitelně záporná. Server DNS A pak výsledek přepošle zpět stanici A – doba zpracování ($RTT1$) tedy bude odpovídat RTT k DNS serveru B. Následně je zjištěno zpoždění k DNS A ($RTT2$), nejlépe iterativním DNS dotazem. Ten je zpracováván pouze na základě výsledků předchozích dotazů a záznamů daného serveru. Zpoždění mezi oběma servery je pak dáno rozdílem obou měření – $RTT1 - RTT2$.

Výhodou této metody je kromě přesnosti i spolehlivost a škálovatelnost – DNS servery jsou jako jedna z klíčových infrastruktur sítě Internet jsou dostatečně dimenzovány, takže případné větší rozšíření této služby nezpůsobí podstatný nárůst provozu a z něho plynoucí zátěže DNS serverů, jelikož většina provozu je důsledkem prohlížení WWW stránek. Přesnost odhadu vyplývá ze sdílení velké části cesty mezi A a B s cestou mezi jejich blízkými DNS servery, nejčastěji se cesta liší na obou stranách v posledních 3 mezilehlých uzlech [22].

3.4 Systém Meridian

Systém Meridian je příkladem moderní metody vyhledání nejbližšího uzlu a nepřímo tedy i predikce zpoždění pro uzly překryvné sítě a také uzly mimo tuto síť, síť uzlů tohoto systému tedy může vytvořit globální službu, o kterou se snažily dřívější pokusy jako např. IDMaps. Struktura systému je velmi volná, netvoří žádnou konkrétní strukturu, čímž je vhodná pro moderní překryvné sítě . Koncepce umožňuje i složené

dotazy s více podmínkami a zjišťovat tak např. uzly, které jsou v dané vzdálenosti od více uzlů zároveň, přičemž vyhledání má výpočetní složitost $O(N)$. Pro toto dohledávání je vždy použita série přímých měření zpoždění, jejich počet je však zřejmě dostatečně malý.

Každý uzel člení prostor kolem sebe do mezikruží s exponenciálně se zvětšujícím vnějším poloměrem, do nichž jsou umisťovány okolní uzly. Počet uzlů umístěných do každého mezikruží je vždy shora omezen. Tím je hustota uzlů umístěných v okolí nejvyšší a každý uzel tak slouží jako jakási místní autorita a odpovídá na dotazy týkající se těchto blízkých uzlů. Zároveň zná zpoždění k stávajícím nejbližším uzlům. Udržuje si však neustále i dostatečný počet uzlů ve vnějších prstencích, kterým může předat dotaz ke zodpovězení.

Pro odhad zpoždění k libovolnému uzlu lze využít proces vyhledání uzlu systému Meridian nejblíže uzlu, ke kterému je třeba znát zpoždění (dále *cílový uzel*, CU) a poté jako odhad použít zpoždění k tomuto uzlu. Ten bude nejbližší několika jiným uzlům, takže dochází stejně jako u předchozích metod k snížení počtu možných dvojic, mezi nimiž se měření provádí.

Hledání je zahájeno zasláním dotazu kterémukoli uzlu sítě Meridian. Tento uzel změří zpoždění d k CU a určí do kterého mezikruží s číslem j CU patří. Dotaz je poté rozeslán všem členům mezikruží j, $j \pm 1$ vzdáleným ± 50 % hodnoty d. Každý z nich změří zpoždění k CU a výsledek vrátí jako odpověď. Dohledání je delegováno na ten z uzlů, který se nejvíce přiblížil k CU. Rekurzivní dotazování končí, pokud žádný z tázaných uzlů nedosáhl přiblížení menšího než prahová hodnota β, což vede k exponenciálnímu přibližování a tím k malému počtu kroků pořebných k dohledání nejbližšího uzlu. Poslední uzel, který byl dohledáním pověřen , je prohlášen za nejbližší. Jeho adresa je postupně předávána tímto řetězem uzlů zpět až do uzlu, který zaslal původní požadavek.

Modifikací tohoto postupu lze pak určit uzel, který je umístěn uprostřed množiny ostatních uzlů T a má tedy nejmenší průměrné zpoždění k ostatním. Mezi aplikace patří například nalezení vhodného herního serveru, zdroje skupinového multimediálního přenosu či reprezentanta shluku uzlů. V tomto případě požadavek na vyhledání obsahuje místo CU množinu T, k jejímž prvkům každý dotazovaný uzel měří zpoždění a jako odpověď vrací průměr těchto zpoždění. Postupné delegování dotazu je pak shodné.

Tento systém lze považovat za perspektivní. Klade minimální nároky na strukturu případné překryvné sítě, která jej bude využívat, má nízké nároky na výpočetní výkon, dostatečně malou režii a velmi malou chybu ve srovnání se systémy využívajícími souřadnicové systémy. Pro síť 2500 uzlů (jejíž měření, označená jako sada Meridian, jsou použita dále v rámci simulací v kapitole 6.2.1) dosahovala o řád menší chyby než GNP a algoritmus Vivaldi, přibližně 1 ms. Pokud by tento systém využila

některá z rozšířených překryvných sítí, vznikla by velmi přesná celosvětová služba predikce vzdálenosti s minimálními požadavky na stanice tuto službu používající.

4 KNIHOVNA VIVALDI-LIB

Jako základ pro simulaci sítí pro tuto práci je použita knihovna vivaldi-lib, vyvinutá Bc. Tomášem Handlem v rámci semestrálního projektu a navazující diplomové práce [25]. Knihovna obsahuje dvě implementace algoritmu Vivaldi – původní jednoduchý algoritmus s konstantním ČK, vyvinutý v rámci semestrálního projektu, a algoritmus s adaptivním ČK, vyvinutý v rámci diplomové práce. Obě knihovny pracují s původním 2D souřadnicovým systémem s výškou, knihovna je však navržena dostatečně obecně a rozšíření tak není problémem. Například lze jednoduchou úpravou předefinovat počet rozměrů eukleidovského prostoru s výškou nebo použít zcela jiný typ souřadnicového systému.

Knihovna byla napsána v programovacím jazyku Java. Tento jazyk byl zvolen jako moderní objektový jazyk. Výhodou objektových jazyků je jednoduché použití již napsaného kódu, což je dáno chápáním objektů jako uzavřených, samostatných jednotek kódu. Dalším významným rysem jazyka Java je nezávislost na platformě – knihovny a programy tedy lze spustit na libovolném operačním systému a tedy i libovolném síťovém uzlu. Předpokladem je pouze přítomnost interpretu zkompilovaných Javovských programů – tzv. Java Virtual Machine (JVM) [27], známý na osobních počítačích jako Java Runtime Environment (JRE).

Oproti druhému z rozšířených systémů s obdobnými prvkem virtuálního stroje – Microsoft .NET Framework – je Javovský virtuální stroj rozšířený i v mobilních telefonech s výjimkou nejnižší cenové kategorie. Pokud by byl implementován kód pro síťovou komunikaci místo prosté evidence metrik (zpoždění) spojů k ostatním uzlům, mohly by tedy uzly sítě Vivaldi být implementovány i jako součást síťových aplikací pro mobilních telefonech.

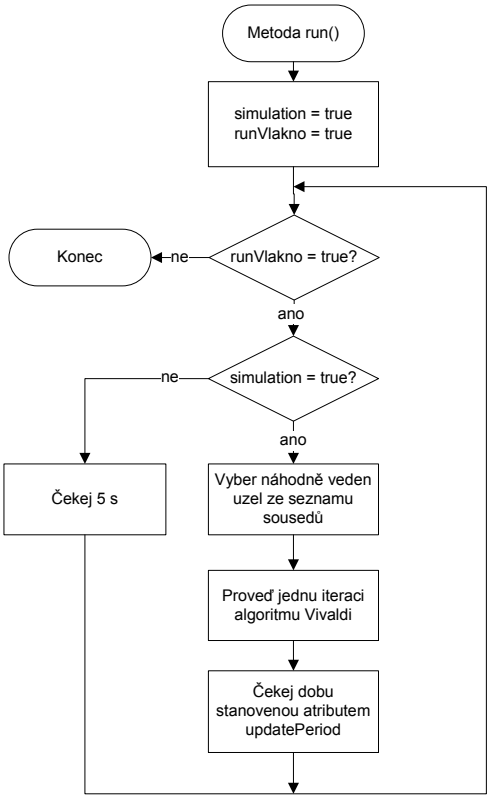
4.1 Knihovna s pevným krokem

Součástí vivaldi-lib je původní knihovna algoritmu Vivaldi s konstantním ČK. Ze sady tříd nabízených knihovnou je pro použití v dále popsaném ovládacím uživatelském rozhraní důležitá zejména třída `Vivaldi_node`. Tato třída je koncipována jako vlákno – implementuje rozhraní `Thread`, což umožňuje algoritmus výpočtu souřadnic spustit na pozadí. To je pro vytvoření simulačního programu velmi důležité, protože může pouze dát pokyn ke spuštění všech simulovaných bodů, nechat simulaci běžet a v případě potřeby se dotázat na zaznamenané hodnoty souřadnic a mezivýsledků každé iterace.

Třída je velmi jednoduchá. Pro simulaci jednoduché statické sítě je třeba volat tři metody - konstruktor, přidání uzlu `addNode()` a spuštění vlákna simulace `run()`. Třída obsahuje i definici jedné iterace algoritmu jako samostatnou metodu

`execute()`, která je definována jako veřejná a lze tedy využít je samostatně bez spuštění vlákna. To je vhodné například pro případné ladění.

Konstruktor třídy `Vivaldi_node2` určuje typ uzlu při jeho vytvoření. Umožňuje určit zejména počáteční pozici uzlu, uzel je dále možné pojmenovat smysluplným jménem pomocí parametru `name`. Přesněji jsou zde konstruktory dva, první bude používán zejména pro testy reálných sítí, jako je např. sady King1 či Meridian, druhý pro umělé sítě. Původní algoritmus inicializuje počáteční souřadnice do počátku souřadné soustavy, tato možnost určit počáteční pozici umožňuje dále experimentovat s vlivem počátečního rozložení bodů na konvergenci algoritmu.



Obr. 4.1: Algoritmus vlákna simulačního uzlu

S dvěma definicemi konstruktoru souvisí i dvě podoby metody `addNode()` pro přidání uzlu – v případě použití prvního konstruktoru se 3 parametry je nutné použít volitelný parametr `rtt` metody `addNode()` (druhá podoba), pokud je použit první konstruktor, můžeme použít obě verze. První verze, mající jediný parametr v podobě známého protějšího uzlu, využije souřadnice určené parametrem `point2` při volání konstruktoru a vypočte RTT na základě vzdálenosti v rovině doplněné o výšku. Tak lze velmi snadno sestavit simulaci umělé sítě pouze nadefinováním souřadnic v jakémsi modelu sítě a jeho protějších uzlů.

Typické použití metody zahrnuje tyto tři kroky v daném pořadí:

1. vytvořit objekt třídy `Vivaldi_node2` voláním konstruktoru třídy,
2. pro každého souseda `u` daného uzlu volat metodu `addNode(u)` nebo `addNode(u, RTT)`,
3. spustit simulaci voláním metody `run()` – její algoritmu viz obrázek 4.1.

Kromě tohoto základního použití je běh vlákna řízen dvěma atributy `simulation` a `runVlakno`, fungujícími jako jakési stavové proměnné. Nastavení `simulation` na `false` způsobí pozastavení simulace, totéž nastavení pro `runVlakno` ukončí smyčku simulace a poté celé vlákno. Celý životní cyklus vlákna je shrnut v příloze A.6. Tento diagram vychází z diagramu v [25], je pouze rozšířen z důvodu lepší orientace o označení logických stavů objektu[1]. Oba cykly v diagramu odpovídají společně hlavnímu cyklu vlákna simulačního uzlu.

Jsou zde definovány i metody pro manipulaci s už definovanými uzly a jejich RTT – `removeNode()`, `updateRTT()`, nejsou však v dále popsaném uživatelském rozhraní použity. Strukturu celé knihovny lze shrnout pomocí UML diagramu tříd v příloze B.

4.2 Pomocné třídy

Pro zaznamenávání okamžitých hodnot důležitých veličin každé instance slouží objekty třídy `NodeRecord`, tyto záznamy uzel ukládá do jemu určeného objektu `NodeHistorieRecord`. Tento objekt je mu přiřazován jako poslední argument konstruktoru uzlu, který nebyl v předchozí části zmíněn. Třída `NodeHistorieRecord` samozřejmě zpřístupňuje všechny vzorky, ale z hlediska rychlosti přístupu jsou prvky uloženy v pořadí *od nejnovějšího* po nejstarší. Pro pohodlí je zde metoda `getLastRecord()` zpřístupňující poslední zaznamenaný vzorek veličin. Všechny záznamy jsou ukládány do paměti, takže při dlouhých simulacích mohou být paměťové nároky poměrně značné. V takovém případě je zřejmě vhodné vytvořit potomky těchto tříd, ukládající záznamy na pevný disk, jelikož disponuje podstatně větším prostorem pro uložení dat simulace.

Souřadnice jsou reprezentovány rozhraním `Coordinate`, které především definuje základní operace v souřadnicovém systému – přičtení, vzdálenost, odečtení, násobení skalární veličinou. Pro změnu souřadnicového systému je tedy potřeba pouze použít objekty jiné třídy než je výchozí třída `Coordinates`.

[1] Logické stavy proto, že jsou určeny pouze hodnotami zmíněných příznaků a tím, zda je vlákno spuštěno či ne.

4.3 Knihovna s adaptivním časovým krokem

Použití této knihovny je velmi podobné, jako u výše uvedené knihovny s krokem adaptivním. V případě vlastních uzlů jsou použity objekty třídy `Vivaldi_node2`, která jsou opět koncipována jako vlákna. Jediným rozdílem z pohledu rozhraní přidání parametru `ei` konstruktoru, představujícího počáteční hodnotu vlastní chyby uzlu e_i. Přibývají také atributy specifické pro konstantní krok, tj. místo atributů `cc`, `ce` je zde pouze atributů delta pro nastavení velikosti ČK. Všechny třídy knihovny a jejich vztahy jsou opět shrnuty v příloze B

Pomocné třídy `Coordinates`, `Coordinate` jsou pro obě knihovny shodné, stejně jako `NodeHistorieRecord` a `NodeRecord`. Třída `NodeRecord` byla pro účely knihovny s adaptivním ČK rozšířena o atributy pro uložení mezivýsledků algoritmu s adaptivním krokem. Jsou však z důvodu kompatibility ve výchozím nastavení nulové a při procházení záznamů algoritmu s konstantním ČK nejsou čteny.

5 VYVINUTÝ SIMULAČNÍ PROGRAM VIVALDIMONITOR

Na simulační knihovnu stručně popsanou v předchozí části navazuje vyvinutá aplikace VivaldiMonitor, tvořící grafické rozhraní pro tuto knihovnu. Cílem vývoje měla být aplikace, která by umožnila pozorovat činnost sítě uzlů implementujících algoritmus Vivaldi jako distribuovanou decentralizovanou vzdálenostní službu. Tato aplikace, jejíž vývoj byl hlavním cílem práce, je poskytnuta ve dvou variantách – hlavní variantou je VivaldiMonitor s proměnným časovým krokem, využívající vlákna `Vivaldi_node2`. Pro účely srovnání pak byla na podobné rozhraní adaptována i původní aplikace Vivaldi GUI pro konstantní časový krok. Obě jsou poskytnuty ve formě spustitelného souboru[1] připraveného k použití bez nutnosti mít nainstalované vývojové prostředí a knihovny. Přiloženy jsou i zdrojové kódy v podobě projektu vývojového prostředí NetBeans IDE verze 6.5.

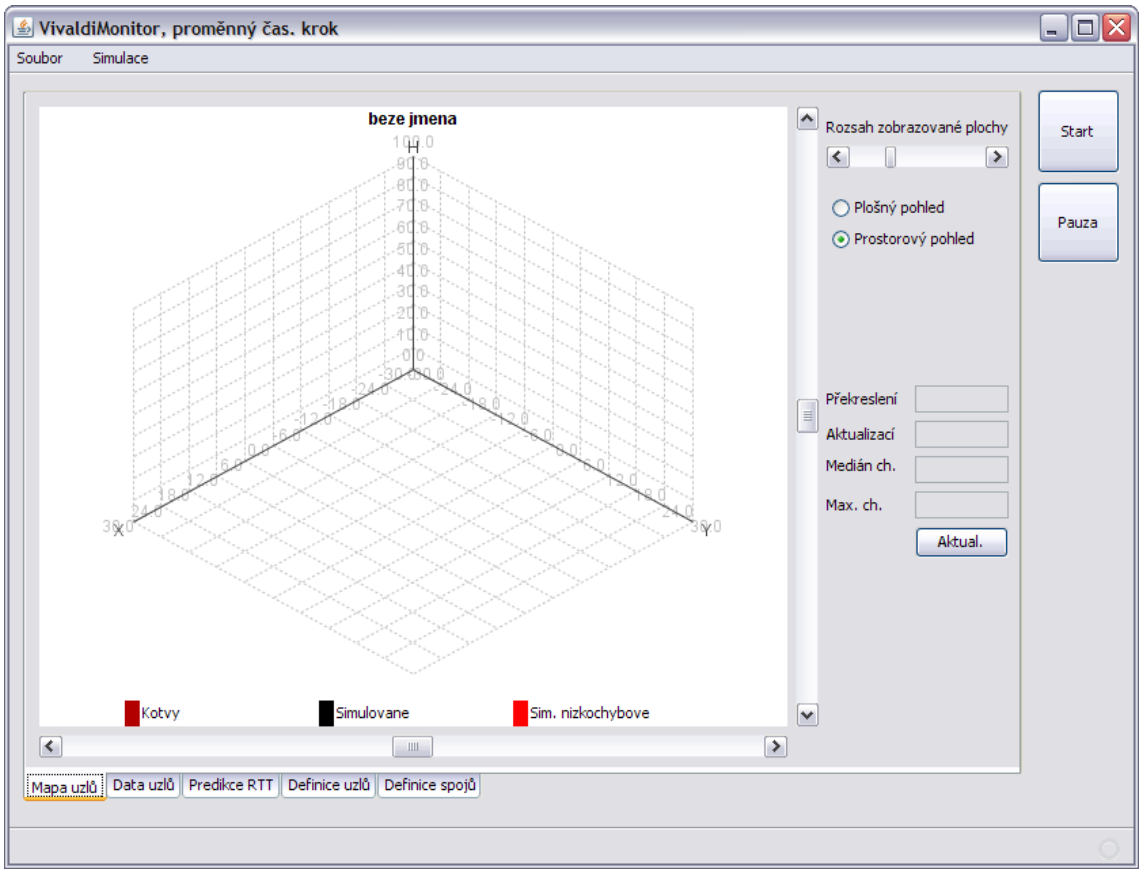
Znázornění činnosti vyžaduje pečlivou volbu veličin, co nejlépe popisující aktuální stav sítě i její vývoj. Z tohoto důvodu aplikace nabízí několik různých pohledů na zaznamenané vzorky veličin uzlů sítě. Jde jmenovitě o zobrazení aktuálních souřadnic uzlů prostřednictvím grafu, výpis aktuálního vzorku údajů všech uzlů v tabulce, dále pak grafické zobrazení předchozího vývoje souřadnic uzlů, časová závislost relativní chyby predikce vzdálenosti a výšky uzlu.

Samozřejmostí je i možnost načtení předpřipraveného modelu sítě ze souboru, jeho úprava a uložení. Je možný i jednoduchý export záznamů jednotlivých uzlů i sítě jako celku v podobě skriptu prostředí Matlab, do souboru lze uložit i přehled spojů společně s jejich predikcemi a chybou predikce.

Aplikace byla napsána v Javě z důvodu přenositelnosti na jiné operační systémy. Jako základ pro aplikaci byl použit aplikační rámec (framework) Swing Application Framework [30], založený na knihovně grafického uživatelského rozhraní Swing [18, 56]. K běhu aplikace je třeba mít nainstalováno běhové prostředí JRE 1.6, pro vývoj aplikace pak vývojové prostředí NetBeans a JDK (Java development kit) 1.6

První část této kapitoly slouží jako jakýsi manuál k vytvořené aplikaci a jeho rozhraní, v druhé části je pak shrnuta architektura aplikace a popsány děje při ovládání simulace – tj. spouštění, pozastavení apod.

[1] Přesněji jde o spustitelný JAR (Java archive) soubor, nejde tedy o klasický spustitelný soubor (přípona `exe`).



Obr. 5.1: Vzhled hlavního okna aplikace VivaldiMonitor

5.1 Popis rozhraní a ovládání

5.1.1 Základní struktura rozhraní

Aplikace je pro jednoduchost ovladatelná z jejího hlavního okna, zachyceného na obr. 5.1. Toto hlavní okno obsahuje sadu několika karet, které velmi efektivně zpřístupňují většinu funkcí. Tato sada karet obsahuje dvojici karet pro zobrazení aktuálního rozložení uzlů probíhající simulace – *Rozložení uzlů* a *Data uzlů*. Za ní následuje karta *Predikce RTT*, která umožňuje zhodnocení přesnosti predikce zpoždění všech spojů. Poslední pak soustřeďuje veškeré rozhraní pro definici topologie sítě. Podrobněji se tomuto rozhraní věnují následující kapitoly formou návodu k použití.

Hlavní rozhraní je doplněno dvěma nabídkami ve standardní nabídkové liště pod titulkovým pruhem okna. Jde především o standardní nabídku *Soubor*, umožňující import a export modelů a dat, a menu *Simulace*, umožňující nastavit parametry simulace a také uložit aktuální obsah grafů rozložení sítě a vývoje predikční chyby jako obrazové soubory.

5.1.2 Načtení hotového modelu

Pokud je síť, kterou je třeba simulovat, dostupná ve formě modelu (souboru s příponou `nm`), lze tento model jednoduše načíst pomocí položky nabídky *Soubor – Načíst model*. Po vybrání položky se objeví standardní dialogové okno pro výběr souboru. Po vybrání souboru a odsouhlasení dialogu je model načten a je možné s ním dále pracovat. V případě rozsáhlejších modelů či méně výkonného počítače je třeba strpení, načtení může chvíli trvat.

V opačném případě, tedy pokud potřebný model dostupný není, existují dvě možnosti – v případě jednodušších sítí lze použít definici topologie pomocí rozhraní aplikace nebo si tento model nechat připravit pomocí k tomu určenému programu. Pro generování jednoduchých umělých sítí ve formě čtvercové mřížky slouží vyvinutý program `ANMGrid`. Program se spouští z příkazové řádky se čtyřmi parametry – jméno výsledného souboru s modelem sítě, počet uzlů podél jedné strany (pro mříž N uzlů je teto hodnota rovna $\sqrt{N}$), zpoždění mezi sousedními rohovými uzly (délka strany čtverce, tvořeného rohovými uzly) a počet sousedů každého uzlu. Chceme-li tedy vygenerovat síť 100 uzlů s 20 sousedy a se stranou čtverce 100, použijeme následující příkaz:

```
java -jar ANMGrid.jar výstupní soubor 10 100 20
```

Program `ANMGrid` byl použit například k vygenerování modelů pro simulace zmíněné v části 6.1. Chceme-li vytvořit z nějakého už existujícího modelu jiný, který obsahuje jen určitý počet sousedů každého uzlu, lze využít program `NMVyberSousedy`. Tento program je volán se třemi parametry: cestou ke zdrojovému modelu, cestu k souboru, kam má být upravený model uložen a konečně počet sousedů každého uzlu. Celý zápis pro ponechání 20 sousedů pak vypadá takto:

```
java -jar NMVyberSousedy.jar zdrojový soubor výstupní soubor 20
```

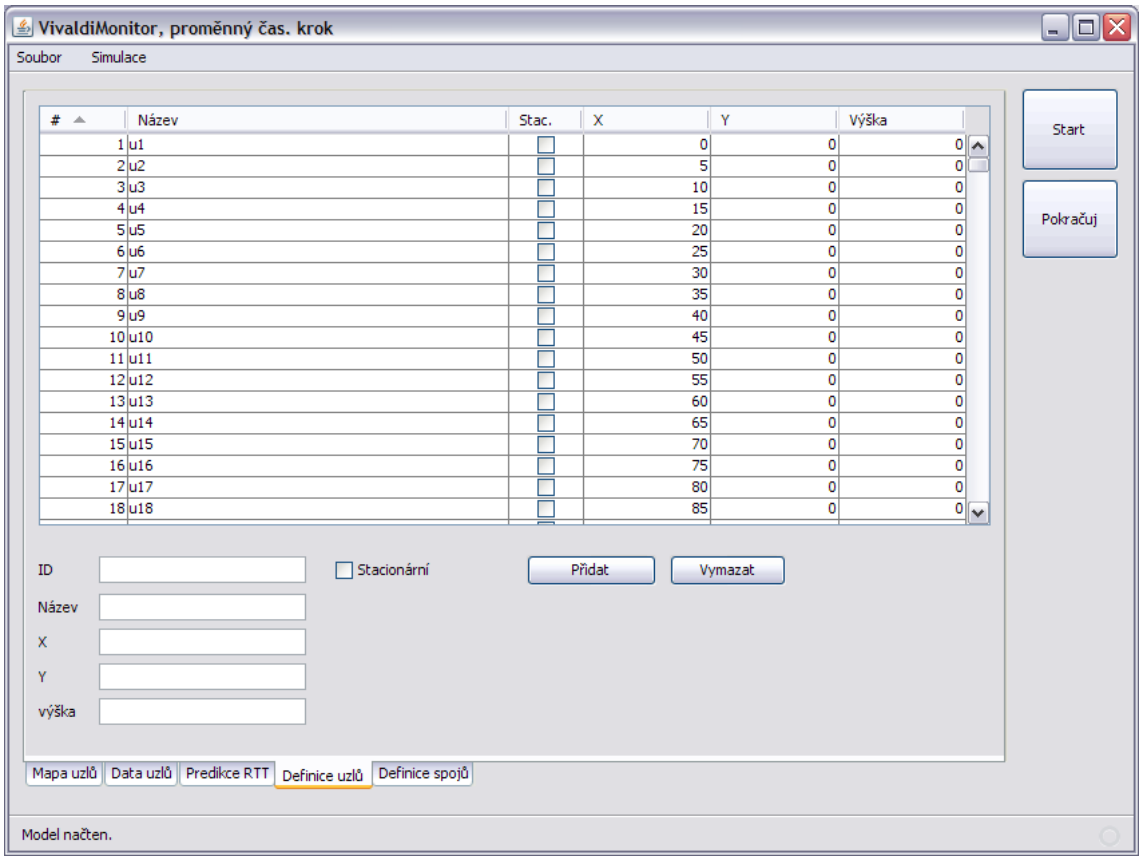
Toto programu lze využít při experimentování se sadami měření z reálné sítě, jako jsou sady PlanetLab, sady měřené Kingovou metodou a podobně. U složitějších případů zbývá pouze automatizovaná příprava pomocí jednoúčelové utility.

5.1.3 Definování a úpravy simulované topologie

Topologii sítě začínáme sestavovat na třetí horní kartě – viz obrázek 5.2, nadepsané *Definice uzlů*. Zde definujeme vlastnosti uzlů sítě, která bude simulována. Nejdůležitější je určení zda jde o stacionární bod[2] či nikoli, a dále souřadnice uzlu. Pro

[2]Stacionární uzly nejsou součástí sítě Vivaldi uzlů, umožňují však působit jako „ukotvení“ pro simulované uzly.

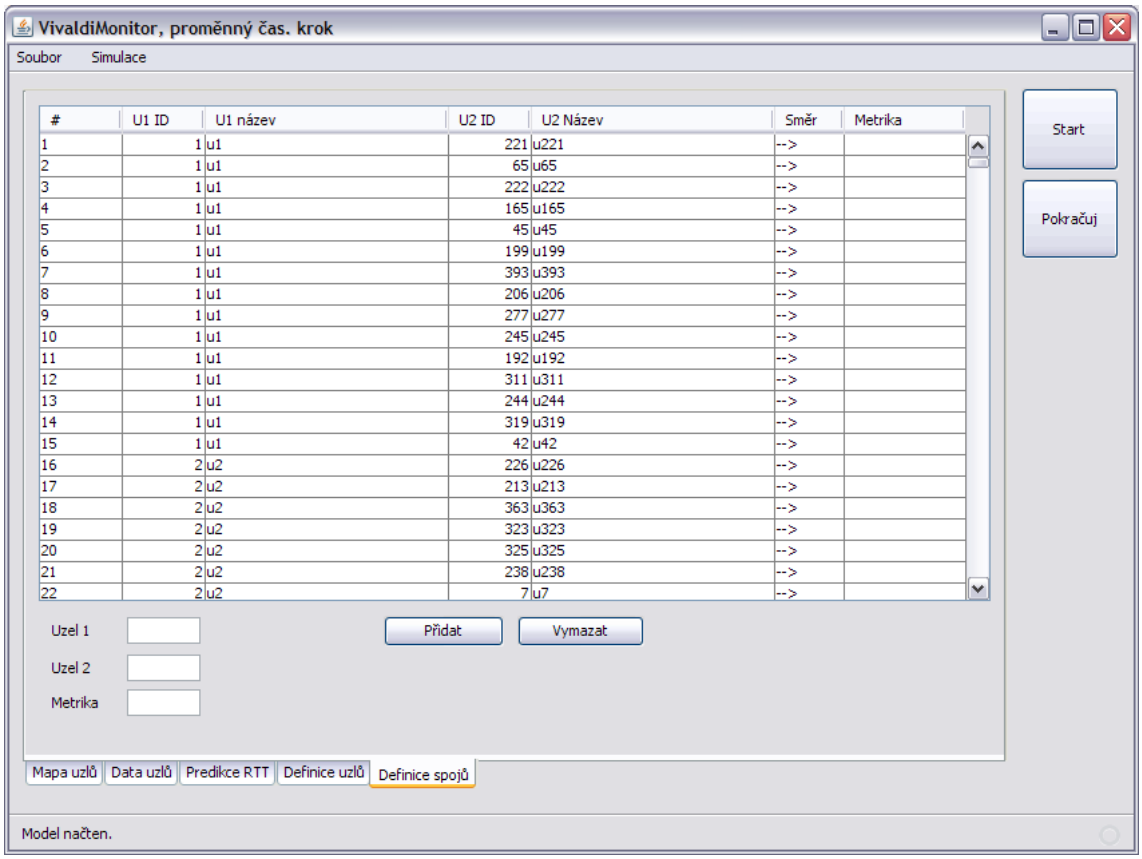
simulované uzly sítě je nutné nechat volbu *Stacionární* nezatrhnutou. Souřadnice uzlu jsou nutnou součástí zadání stacionárního uzlu – určují jeho polohu. U simulovaných uzlů mají význam pro simulaci umělých topologií, kde je možné výpočty vzdáleností mezi uzly určit automaticky, viz dále u definic spojů mezi uzly.



Obr. 5.2: Vzhled hlavního okna aplikace VivaldiMonitor – karta *Definice uzlů*

První dva údaje – pořadové číslo a jméno – simulaci nijak neovlivňují a slouží čistě k pozdější identifikaci uzlů v průběhu simulace a v tabulce spojů. Pořadové číslo je vlastností každého uzlu, jméno lze ponechat prázdné. U těch uzlů, u kterých bude pravděpodobně třeba zobrazovat historii vývoje souřadnic, je vhodné ho zadat, objeví se pak jako popisek v legendě grafu. Souřadnice uzlu jsou celkem tři, X, Y a výška, odpovídající použitému souřadnicovému systému. Uzel přidáme stiskem tlačítka *Přidat*. Úprava údajů je rovněž možná přes tuto tabulku, na danou buňku tabulky je před úpravou třeba poklepat. Úprava je provedena po přesunu na jinou buňku. Odebrání provedeme výběrem odpovídajícího řádku kliknutím v tabulce nad polem pro přidání uzlu a stiskem klávesy *Delete*. Pokud je odebírán uzel poté, co už byly také nadefinovány nějaké uzly, je třeba nejprve ověřit, že nee xistuje žádný spoj odebíraného uzlu s uzlem jiným.

Po nadefinování uzlů definujeme spoje mezi uzly pomocí formuláře na následující



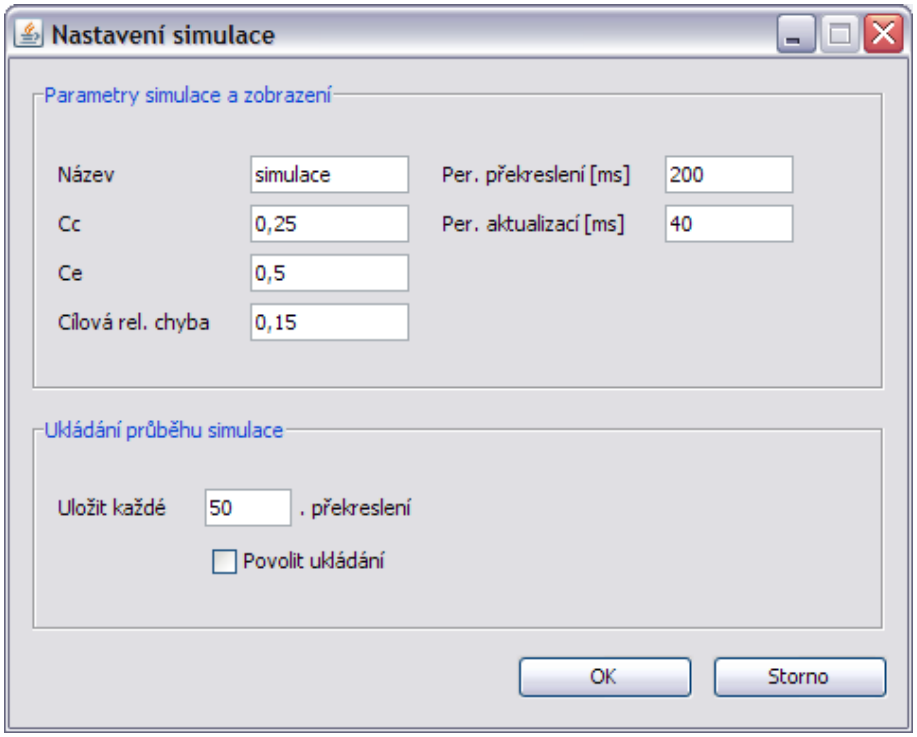
Obr. 5.3: Vzhled hlavního okna aplikace VivaldiMonitor – karta *Definice spojů*

kartě *Definice spojů*, viz obrázek 5.3. Vždy zadáme pořadová čísla (*ID*) dvou uzlů, které mají být spojeny. Lze také zadat metriku spoje, která může představovat např. RTT změřené mezi uzly. Zadání metriky je však nepovinné. Není-li zadána, aplikace se ji pokusí dopočítat na základě dříve udaných souřadnic obou uzlů. Přidání a odebrání spoje je obdobné přidání uzlu. Při kontrole zadaných spojů se lze řídit podle názvů uzlů, které jsou společně s čísly uzlů (zadanými při vytvoření či úpravě spoje) zobrazeny v přehledové tabulce.

5.1.4 Spuštění a ovládání simulace

V této fázi je síť připravena k simulaci. Před vlastní simulací ještě třeba zkontrolovat a případně změnit parametry simulace. K tomu slouží dialog *Nastavení simulace* – viz obrázek 5.4, Základními parametry jsou konstanty c_c, c_e, jimž odpovídají políčka *Cc*, *Ce*. Dále lze nastavit periodu aktualizací a překreslování. Perioda aktualizace je udána v milisekundách a určuje tak rychlost simulace. Vzhledem k asynchronnímu běhu každého jednotlivého uzlu je průměrná perioda aktualizace pro simulaci N uzlů jako celek Nkrát menší. V milisekundách je udána i perioda překreslení. Pro základní

simulace doporučuji ponechat výchozí nastavení, zejména periodu překreslení není vhodné nastavovat níže kvůli rychlosti překreslování grafů, zvláště u větších sítí.



Obr. 5.4: Aplikace VivaldiMonitor – Vzhled dialogu *Nastavení simulace*

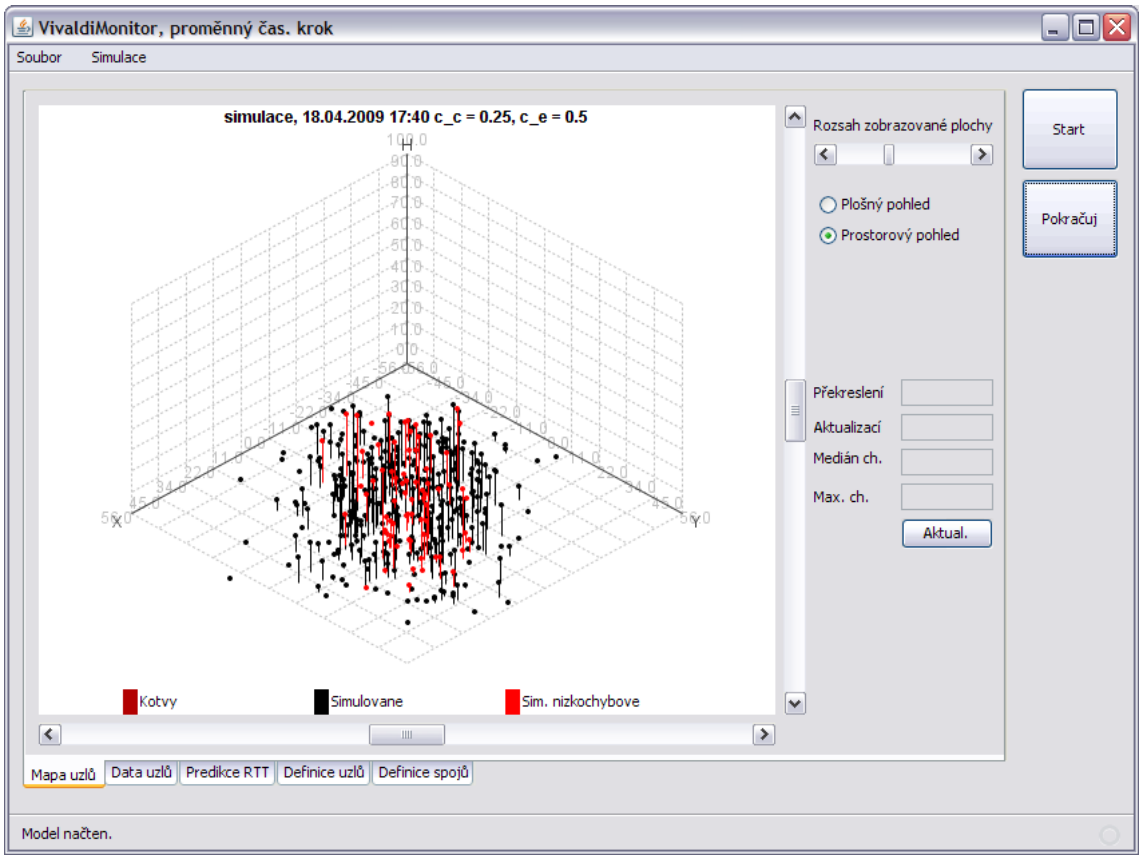
Samozřejmě je možné nastavit název simulace, který se objeví i v nadpisech grafů a umožní tak identifikaci snímků dané simulace v případě, že v jednom adresáři budou snímky více simulací. Posledním nastavením je možnost zachytávání snímků v průběhu simulace.

Průběh lze sledovat na některé ze dvou karet *Mapa uzlů* a *Data uzlů*, aplikace na ně však automaticky nepřepíná a je tedy nutné přepínat je ručně. Spouštění a zastavování simulace je intuitivní, tlačítkem *Start* je spuštěna zcela nová simulace se zadanými parametry a topologií, druhé tlačítko – *Pauza* resp. *Pokračuj* pak slouží k pozastavení resp. k pokračování v simulaci. Příklad grafu rozložení uzlů v průběhu simulace je zobrazen na obrázku 5.5.

Průběh simulace je vzorkován standardně každých 200 ms a vzorek pozice každého uzlu se vykreslí do grafu na první horní záložce – *Mapa uzlů*, na druhé pak ostatní proměnné a mezivýsledky, které jsou zmíněny níže.

5.1.5 Přístup k průběhům proměnných uzlů a sítě

Na této záložce jsou dále dostupné dvě další funkce – zapnutí vykreslení dosavadního vývoje souřadnic vybraných uzlů a zobrazení dalších grafů k vybranému uzlu.



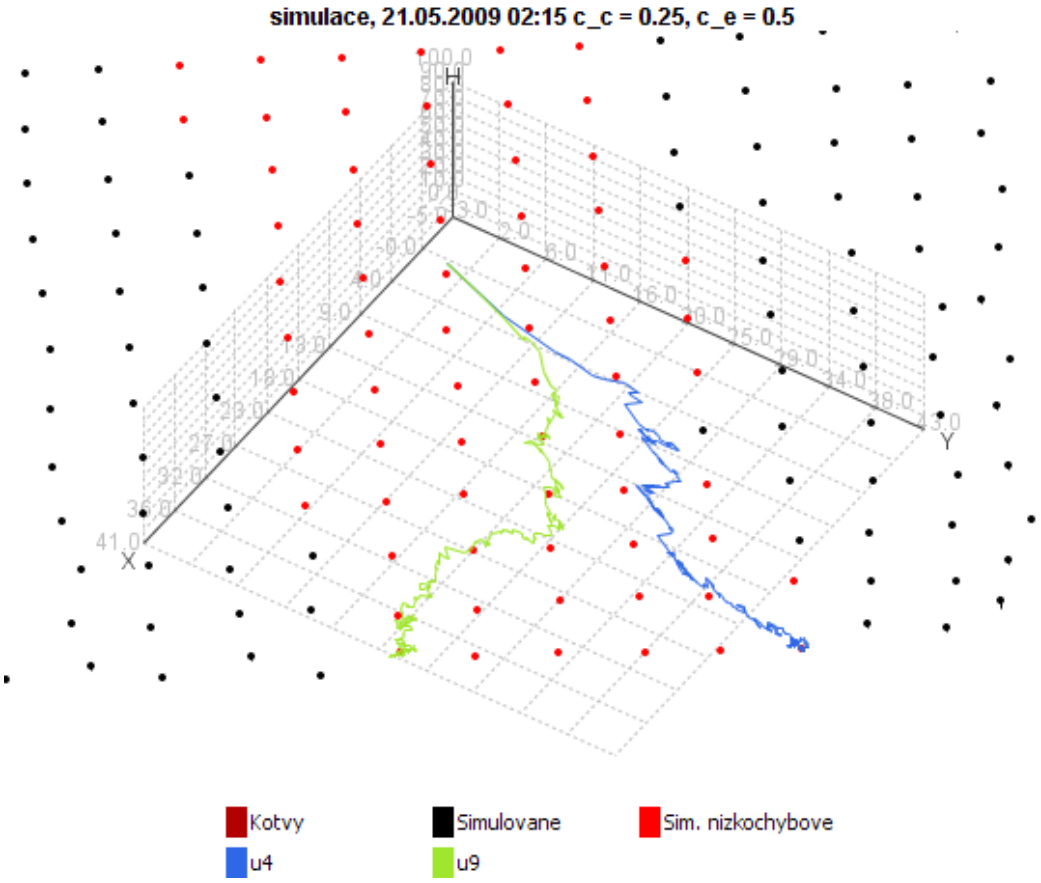
Obr. 5.5: Vzhled hlavního okna aplikace VivaldiMonitor během simulace

Dosavadní vývoj začne být vykreslován u uzlů vybraných v tabulce na záložce *Data uzlů*. Více uzlů lze vybrat známým způsobem – v případě za sebou jdoucích řádků kliknutím na první řádek výběru a následným kliknutím na posledním s přidrženou klávesou *Shift*, případně lze vybrat několik současně, je-li držena klávesa *Ctrl*. Po spuštění nebo při pokračování v simulaci se vykreslí dosavadní posuny a při každém překreslení se v souladu s vývojem simulace doplňuje. Příklad je zachycen na obrázku 5.6.

Průběhy důležitých proměnných lze jednoduše zobrazit u každého uzlu poklepáním na odpovídající řádek tabulky souřadnic uzlů na této záložce. Poklepáním je vyvoláno dialogové okno *Vývoj uzlu* (příklad je na obrázku 5.7), obsahující v horním grafu průběhy absolutní hodnoty chyby predikce grafu a chyby uzlu e_i a dále v dolním grafu průběh časového kroku a výšky uzlu.

V dolní části karty *Data uzlů* se dále nachází graf vývoje chyby sítě – viz obrázek 5.8. Tento graf obsahuje průběhy pro 10 a 80. percentil[3], dále medián (tj. 50. percentil) a maximum. Lze tak zhodnotit charakter rozložení chyby predikce. Důležitým ukazatelem je především medián a dále 80. percentil, které se pro zhodnocení

[3]Definice percentilu viz kapitola 6.2.2.



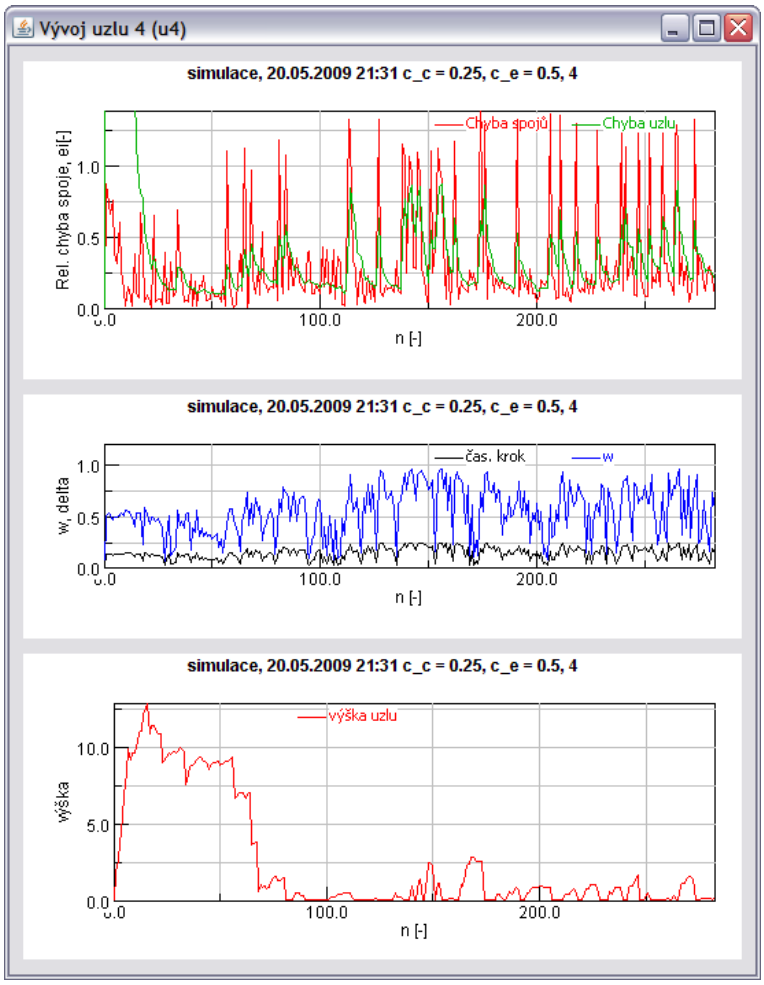
Obr. 5.6: Vzhled hlavního okna aplikace VivaldiMonitor během simulace – stopa vývoje uzlu

přesnosti predikce nejčastěji používají. Tyto ukazatele nahrazují částečně histogram, jehož vývoj v čase nelze dobře zobrazit.

5.1.6 Zhodnocení predikce RTT spojů

Kdykoli po spuštění simulace lze provést analýzu kvality predikce spojů. K tomu je určena karta *Predikce RTT*, v pořadí třetí – viz obrázek 5.9. Hodnotit lze predikci jak spojů definovaných v modelu, tak i spojů, které v aktuálním modelu obsaženy nejsou. Podmínkou hodnocení spojů, které v aktuálním modelu nejsou, je dostupnost jiného modelu, který obsahuje všechny uzly aktuálního modelu a větší počet spojů, než má aktuální model – tzv. referenční model. Nejčastěji je k dispozici model sítě, tvořící úplný graf a aktuálně simulovaný je připraven odebráním části spojů. To je i případ všech modelů použitých pro simulace popisované v kapitole 6.

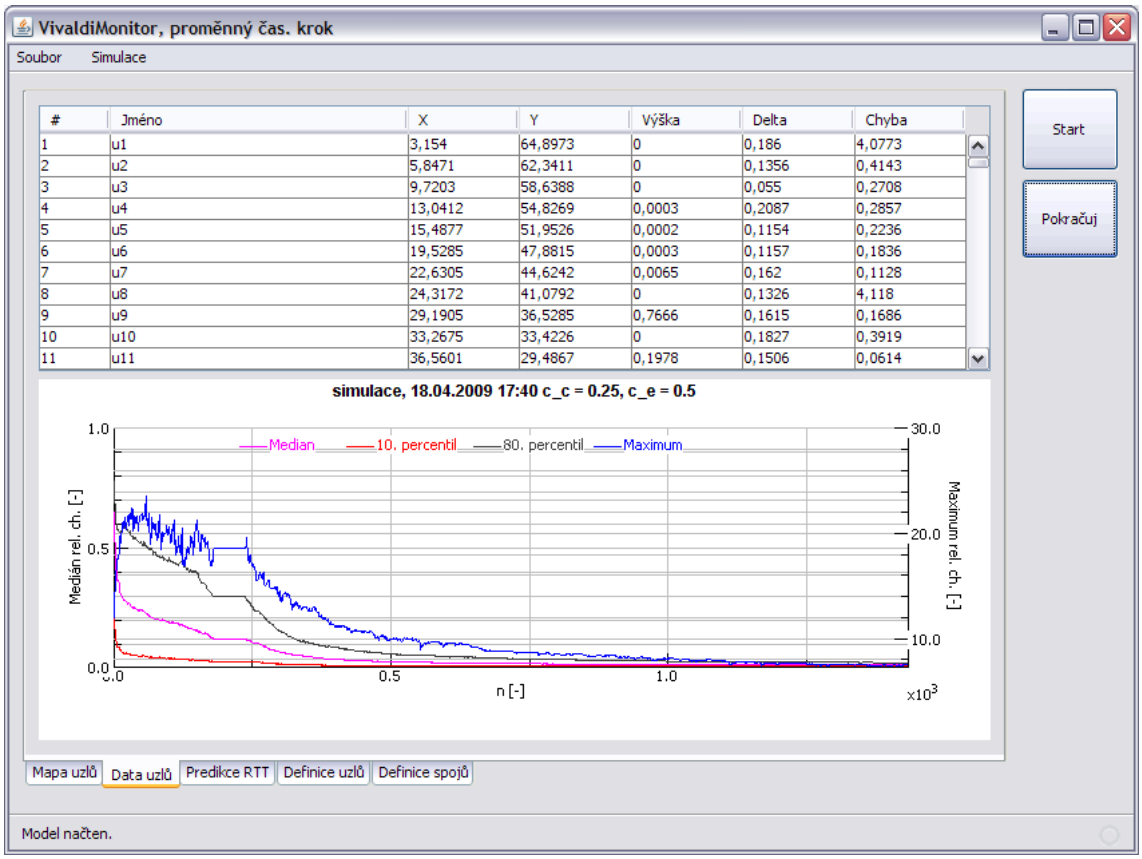
Karta *Predikce RTT* obsahuje dvě podobné skupiny prvků. Horní slouží pro zhodnocení predikce známých spojů, dolní pak pro zhodnocení predikce spojů v aktuálně simulovaném modelu neobsažených („neznámých“) vůči referenčnímu mo-



Obr. 5.7: Aplikace VivaldiMonitor – vývoj proměnných uzlu během simulace

delu. Vlastní porovnání se u obou případů provede po stisknutí tlačítka *Porovnat*. Výsledkem porovnání je tabulka s výpisem všech známých, resp. neznámých spojů a jejich údajů a dále základní statistické ukazatele vypočtené z těchto dat. V případě hodnocení predikce neznámých spojů je třeba předem (nejlépe před vlastními simulacemi) načíst referenční model. Ten lze zvolit pomocí dialogu vyvolaného tlačítkem *Zvolit ref. model*.

Pokud je třeba provést další analýzu kvality predikce, lze využít možnosti uložení získaných dat do souboru, k čemuž je určeno tlačítko *Uložit*, nacházející se rovněž pod každou tabulkou. Výsledný soubor je jednoduchý textový soubor obsahující záznamy z tabulky, jednotlivé hodnoty v řádku jsou odděleny pomocí tabulátoru. Tento formát lze snadno načíst všemi moderními tabulkovými kalkulátory a dále zpracovávat.



#	Jméno	X	Y	Výška	Delta	Chyba
1	u1	3,154	64,8973	0	0,186	4,0773
2	u2	5,8471	62,3411	0	0,1356	0,4143
3	u3	9,7203	58,6388	0	0,055	0,2708
4	u4	13,0412	54,8269	0,0003	0,2087	0,2857
5	u5	15,4877	51,9526	0,0002	0,1154	0,2236
6	u6	19,5285	47,8815	0,0003	0,1157	0,1836
7	u7	22,6305	44,6242	0,0065	0,162	0,1128
8	u8	24,3172	41,0792	0	0,1326	4,118
9	u9	29,1905	36,5285	0,7666	0,1615	0,1686
10	u10	33,2675	33,4226	0	0,1827	0,3919
11	u11	36,5601	29,4867	0,1978	0,1506	0,0614

Obr. 5.8: Vzhled hlavního okna aplikace VivaldiMonitor během simulace – data sítě

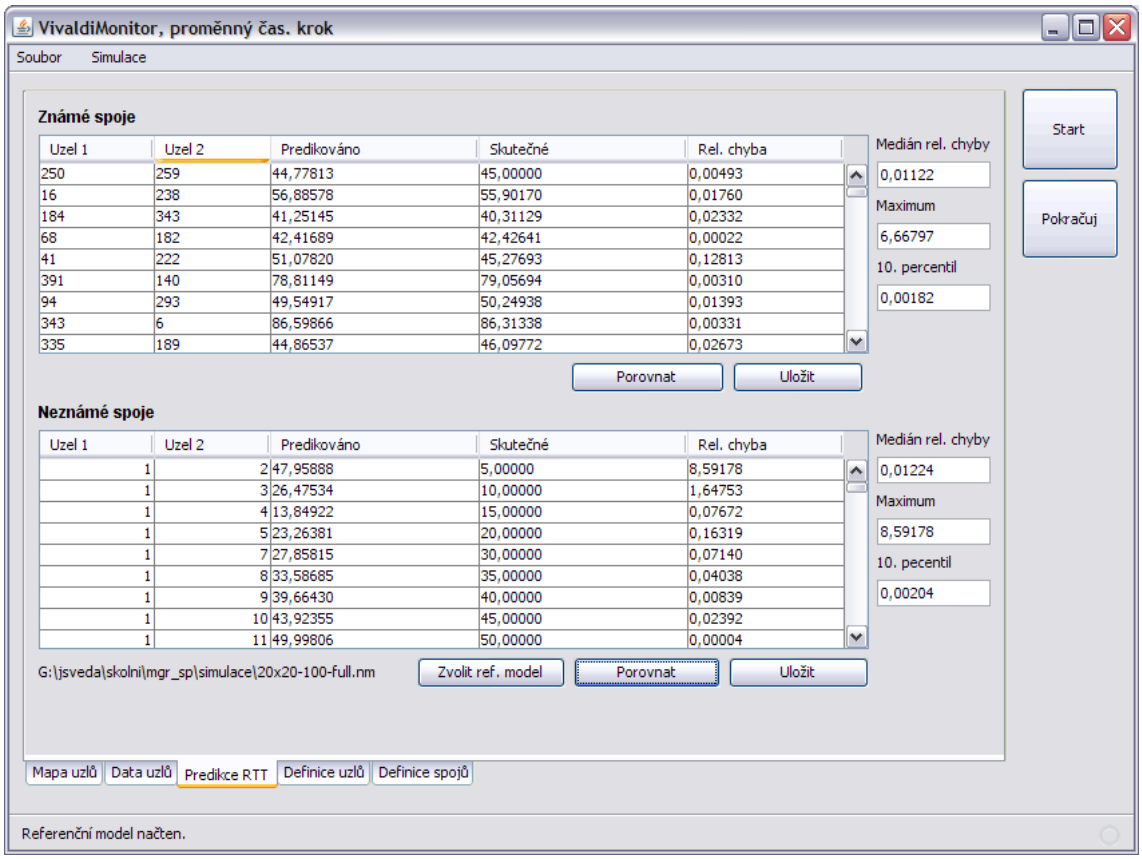
5.1.7 Uložení dat simulace

Data průběhu simulace, dostupné v aplikaci pouze pomocí jednoduchých grafů, lze exportovat do skriptu prostředí Matlab. K tomu slouží položka menu *Soubor – Uložit data simulace*. Vytvořený soubor obsahuje následující proměnné:

- `nhist` – pole záznamů (struktur prostředí Matlab) se záznamy uzlů,
- `nethist` – struktura s daty vývoje chyby,
- `nodenames` – pole řetězců s názvy uzlů pro legendu apod.

Pole `nhist` obsahuje záznamy souřadnic uzlu (`x`, `y`, `h`), dále relativní chybu v dané iteraci zpracovávaného spoje `rel_err` a konečně pořadí iterace `time`. Drobnou úpravou metody `saveSimDataToMFile()` v třídě `VivaldiSimulation` je možné ukládat i ostatní veličiny. Vzhledem k množství dat ukládaných při delších simulacích by si tato úprava vyžádala i možnost volby, které záznamy mají být exportovány.

Pole `nethist` pak obsahuje záznamy vývoje chyby sítě, zobrazené v grafu vývoje chyby. Jedná se tedy o záznamy mediánu (prvek `median_err`), maxima (prvek



Obr. 5.9: Vzhled hlavního okna aplikace VivaldiMonitor během simulace – karta *Predikce RTT*

`maximum_err`), 10. a 80. percentilu (`10pctl_error` a `80pctl_error`). I zde se zaznamenává číslo vzorku v položce `time`, vzorkovací perioda je pevně nastavená na 40 ms. Časový údaj však není ve většině případů příliš důležitý a slouží jednoduše jako čítač, umožňující správné vykreslení grafu.

Jednoduchý příklad zobrazení exportovaných dat představuje skript `zobraz_data_simulace.m`. Je z něho zejména patrné, jakým způsobem lze přistupovat k položkám polí v strukturách `nhist` a `nethist`, který není úplně triviální.

Kromě uložení do skriptu prostředí Matlab, umožňující složitější zpracování naměřených dat, lze uložit i okamžitý obsah obou hlavních grafů v aplikaci, tj. grafu rozložení uzlů a vývoje chyby sítě. K tomu slouží položky menu *Simulace – Snímky*, ukládající grafy jako bitmapové obrazy formátu PNG (portable network graphics). Výhodou je především okamžitá dostupnost těchto grafů, není třeba dalšího zpracování. Pro jakékoli další srovnání je však vhodnější použít zpracování v Matlabu, jelikož použitá knihovna pro grafy má velmi omezené schopnosti.

5.2 Přehled struktury aplikace

Zdrojový kód aplikace je rozdělen do několika hlavních celků, reprezentovaných tzv. balíčky. Na nejvyšší úrovni jde o balíčky

- `cz.vutbr.feec.utko.xhandl02.vivaldi_lib`,
- `cz.vutbr.feec.utko.xsveda07`,
- `au.usq.sci.graph`,
- `org.math.plot`,
- `org.jdesktop`, `org.jdesktop.application`

Vlastní kód aplikace je obsažen v prvním z balíčků, který je dále strukturován do několika částí, které budou uvedeny dále. Zbylé balíčky pak obsahují podpůrné knihovny. Balíček `xhandl02.vivaldi-lib` obsahuje už popisovanou simulační knihovnu, implementující algoritmus Vivaldi v podobě souběžně vykonávaných vláken. Balíčky `au.usq.sci.graph` a `org.math.plot` jsou knihovny pro tvorbu grafů. První je jednoduchá knihovna pro vytváření dvourozměrných čárových grafů [7]. V aplikaci je použita pro vytvoření grafů vývoje chyby a průběhů proměnných uzlu. Druhá, oficiálně nazvaná `JMathPlot` [8], umožňuje kreslení dalších typů grafů včetně histogramů a funkcí dvou proměnných. V aplikaci je pak použit prostorový stopkový graf (třída `BarPlot3D`) a dvourozměrný bodový graf (třída `ScatterPlot2D`). Poslední dva balíčky jsou pak součástí knihovny Swing a souvisejícího prostředí Swing Application Framework, poskytující prvky grafického rozhraní a podpůrné třídy.

Balíčky v `cz.vutbr.feec.utko.xsveda07` jsou rozčleněny následovně:

- `app` – grafické rozhraní aplikace,
- `io` – filtry typů souborů pro dialogy a pomocná třída `SerDes` pro ukládání (serializaci) objektu do souboru,
- `netwmodel` – třídy abstraktního modelu sítě,
- `sim.vivaldi` – jednotné ovládací rozhraní por simulační vlákna a související objekty,
- `vgui.plot` – abstrakční rozhraní pro knihovny pro tvorbu grafů,
- `vgui.plot.graph2d` – adaptace `au.usq.sci.graph` na komponenty knihovny Swing.

Grafické rozhraní je tvořeno třídami formulářů `VivMonitorView` (hlavní okno), `frmHistorieBodu` (okno s průběhy proměnných uzlu) a `frmSimSettings` (nastavení simulace). První dvě třídy pro svou činnost dále využívají tříd pro kreslení grafů přes balíček `vgui.plot`, z nichž nejdůležitější je třída `Graph2D` a rozhraní `MainGraphicViewport` s jejími potomky. U třídy `Graph2D` poznamenávám, že jde o shodné rozhraní jako je `Graph2D` z balíčku `au.usq.sci.graph`, jen používá odlišnou grafickou knihovnu. Původní knihovna používá starší knihovnu AWT, která může být problematická při složitějších grafech vzhledem k absenci techniky tzv. *double-bufferingu*. Mezi nepříjemné projevy vykreslování složitější animace bez jeho použití patří zejména poblikávání vykreslovaného obrazu. Double-buffering používá například použitá knihovna Swing, upravená třída používá jako základ její prvek rozhraní nazvaný `JPanel`. Tato komponenta je běžně používána jako kontejner pro další prvky rozhraní, v tomto případě je použita jako kreslící plátno.

5.3 Charakteristika vybraných tříd

Důležité třídy tvořící aplikaci jsou zachyceny na diagramech tříd v příloze A O třídách tvořících grafické rozhraní je zmíním velmi krátce, jelikož až na hlavní okno `VivaldiMonitorView` jsou triviální. Tyto třídy – hlavní spustitelná třída `VivaldiMonitorApp`, hlavní okno `VivaldiMonitorView` a dva pomocné formuláře `frmHistorieBodu`, `frmSimSettings` – jsou vytvořeny z části pomocí automaticky generovaného kódu vygenerovaného při vytváření projektu a rozhraní ve vývojovém prostředí NetBeans. Třída `VivaldiMonitorView` je poněkud rozsáhlá, představuje však především jen poměrně tenkou „slupku“ umožňující ovládat hlavní aplikační třídy `VivaldiSimulation` a `NetworkModel` a zobrazovat jejich stav a obsah. Tak je možné poměrně jednoduše změnit podobu aplikace bez značných změn ve struktuře výkonných tříd tvořících základ aplikace [48].

Z důvodu oddělení logiky byly přesunuty i některé prvky, zejména objekt simulačního rozhraní a objekt referenčního modelu z třídy hlavního okna do `VivaldiMonitorApp`. Tento referenční model je načten v jedné z pomocných tříd, reprezentujících operace spouštěné v rámci aplikace na pozadí a poté použit při výpočtech chyb predikce po nedefinované spoje.

Základ pro definici simulované sítě tvoří třída `NetworkModel` (viz příloha A.4), která udržuje definované množiny uzlů (atribut `nodes`) a spojů (atribut `links`), umožňuje tyto množiny měnit a vyhledávat v nich, zejména na základě pořadového čísla (ID). Dohromady tak tvoří graf, modelující simulovanou síť. To je zřejmé i z podoby abstraktního spoje `NetworkModelAbstractLink`, který udržuje odkazy na oba uzly, čímž je spojuje. Kromě poskytování rozhraní k těmto dvěma množinám

další funkci neposkytuje. Důležitá je především jako prostředek definování modelu sítě, který může být využit beze změny na více typech simulací – například můžeme simulovat síť, ve které bude mnoho uzlů vzájemně propojených ale ze všech definovaných spojů přiřadíme pro účel simulace jen některé z dostupných spojů.

Kromě pohledu na síť uzlů orientovanou na spoje je také možné spoje k danému uzlu vždy dohledat přes množinu spojů k sousedům objektu třídy `NetworkModelNode` představujícího uzel. Lze tak i plnohodnotně procházet celý graf, jeli třeba (např. pro implementaci Dijkstrova algoritmu). Možnost procházení grafu však v programu využita není.

Objekty třídy `NetworkModel` jsou využity také v souborech modelů. Každý soubor modelu obsahuje právě jeden uložený (serializovaný) objekt této třídy. Lze tak jednoduše pomocí jednoduchých programů v Javě s těmito modely pracovat či generovat nové – např. zpracováním souborů s naměřenými maticemi latencí nebo umělých modelů.

V tomto případě model sítě – tedy třídu `NetworkModel` – využívá třída `VivaldiSimulation`. Její struktura a vazby na ostatní třídy v rámci projektu je zachycena v příloze A.3, vazby na knihovnu vivaldi-lib pak v příloze A.5. Tato třída má dvě hlavní funkce. První funkcí je ovládání simulace – spuštění, pozastavení, pokračování v simulaci – jako celku, která je vnitřně tvořena sadou nezávisle běžících vláken, viz kapitola 4. Tyto funkce jsou realizovány metodami `run()`, `pause()` a `resume()`. Druhou funkcí je zpřístupnění hodnot uložených každou iterací uzlu – tj. „historii“ uzlu. Tyto údaje v podobě objektu třídy `NodeHistoreRecord` jsou přístupné pomocí dvojice metod `getLog()`, `getLogById()`. Pro účely zobrazení hodnot posledně proběhlé iterace každého uzlu je implementována také metoda `getLastNodeRecords()`.

Součástí třídy je definice časovače `calcNetMedianErrTimer`, který provádí každých 40 ms [4] analýzu relativních chyb všech spojů pro účely statistického vyhodnocení. Tento postup je nutný vzhledem k nepřítomnosti mechanizmu upozorňujícího na dokončení iterace simulačního vlákna. Toto upozorňování, provedené voláním k tomu určené metody objektu `VivaldiSimulation`, by umožnilo jednoduše znovu vyhodnotit a zaznamenat sledované ukazatele. Mezi vybrané ukazatele je zařazeno maximum, medián, 10. a 80. percentil[5].

Značné zjednodušení inicializace tříd `Vivaldi_Node` umožňuje třída `ModelSimAssociation`, která při vytváření automaticky nastaví počáteční hodnoty atributů simulačních uzlů a vytvoří i objekt `NodeHistorieRecord`, vyžadovaný konstruktorem simulačního uzlu – třídy `Vivaldi_node`. Objekt této třídy je tvořen trojicí objektů tříd `NetworkModelNode`, `Vivaldi_node` a `NodeHistorieRecord` a provazuje

[4] Tato hodnota nemusí být přesně dodržena, zejména je-li počítač, na němž je aplikace spouštěna, vytížen jinou úlohou.

[5] definice percentilu viz. kapitola 6.2.2

tak model s objekty knihovny.

Z důvodu snadné modifikace počátečních podmínek simulace je také vytvořena abstraktní třída `SimNodePlacementPolicy`, třídy z ní dědící pak definují jaké souřadnice se mají na počátku uzlům přiřadit. Z podobného důvodu je definována i třída `VivaldiParamters` zmíněná dříve. Tato třída umožňuje centrálně definovat parametry simulace.

Oproti původní aplikaci Vivaldi GUI vyvinuté v rámci semestrálního projektu [57] byl oddělen z kódu hlavního formuláře (tj. třídy `VivaldiMonitorView`) kód pro zobrazení grafu rozložení uzlů. Ten je pak soustředěn v třídě `MainGraphicViewport` a jejích potomků (`MainGraphicViewport2DVar1`, `MainGraphicViewport3D`). Potomci `MainGraphicViewport` odpovídají dvěma typům pohledu na rozložení uzlů – plošné (bez vyjádření výšky) a prostorové. Samotná třída `MainGraphicViewport` pak obsahuje společnou logiku, zejména metody volané v reakci na změnu mezních hodnot pomocí posuvníků kolem vlastního grafu.

5.4 Děje při spuštění simulace a v jejím průběhu

Spuštění simulace je možné pomocí volání metody `run()` instance třídy `VivaldiSimulation`. Tato metoda (viz vývojový diagram v příloze A.1) především zastaví všechny případná spuštěná vlákna z případné předchozí simulace a provede úplnou inicializaci všech objektů pro následující simulaci, jelikož se může jednat jak o opakování téže simulace (např. s jinými parametry), tak o simulaci zcela novou. Proto jsou po zastavení vláken zahozeny existující objekty `ModelSimAssociation` a je sestavena nová množina těchto objektů na základě daného modelu sítě. Vytvořením instance třídy `ModelSimAssociation` jsou současně vhodně inicializovány objekty samotného simulovaného uzlu a jeho záznamů.

Při této inicializaci však nejsou objektům třídy `Vivaldi_node2` přiřazeny údaje o spojích k sousedům. To je úkol hlavní smyčky metody. Pro daný uzel modelu je postupně procházena jeho tabulka spojů k sousedním uzlům a každý údaj souseda a vzdálenosti (k němu pokud je definována) je zapsán do množiny sousedů simulačního uzlu.

Po této inicializaci jsou nastaveny konstanty c_c a c_e pomocí odpovídajících statických vlastností třídy `Vivaldi_node2` a všechna vlákna jsou spuštěna, nejde-li o stacionární uzly. Posledním krokem je vytvoření polí pro vzorky sledovaných statistických ukazatelů pro množinu relativních chyb spojů a provedení kroků pro pokračování simulace, z nichž je nejdůležitější spuštění časovače `calcNetMedianErrTimer`, zodpovědného za ukládání pravidelných vzorků hodnot těchto statistických ukazatelů.

Po inicializaci a spuštění vlastní simulace je ještě v hlavním formuláři `VivaldiMonitorView` spuštěn časovač zajišťující překreslení hlavního grafu s rozložením uzlů, grafu vývoje chyby a tabulky s daty simulovaných uzlů. Tento časovač je ve výchozím nastavení spouštěn s periodou 200 ms.

Toto překreslování zajišťuje metoda `run()` třídy `RedrawNetwGraphTask`. V této metodě jsou ze záznamů simulovaných uzlů načteny poslední zaznamenané hodnoty. Ty jsou následně uloženy do množiny objektů třídy `SimViewNodeData`, které tvoří jednotný datový typ pro data zobrazovaná objektem třídy `MainGraphicViewport`.

Objekt `SimViewNodeData` nese i informaci, zda se daný uzel považuje za uzel s nízkou chybou. Jako hodnotící kritérium je zvolen aritmetický průměr posledních $2n$ vzorků relativní chyby predikovaných spojů, kde n je počet sousedů uzlu. Za nízkochybový je uzel považován, pokud daný průměr poklesne pod hodnotu určenou nastavením `VivaldiParameters.targetError`. Tato operace někdy trvá déle, než perioda iterací simulovaných uzlů. Aby bylo možné spolehlivě projít potřebný počet vzorků, čítač indexu vzorku je doplněn o údaj ofsetu, který je zvýšen, kdykoli se v průběhu procházení zvětší počet položek v poli záznamů.

Po naplnění pole pro grafy je aktuálně zvolený typ grafu překreslen s použitím nových dat (pomocí metody `displaySimData`). Je-li zvoleno ukládání snímků tohoto grafu, je nový obsah grafu rovněž uložen. Následuje aktualizace tabulky dat uzlů a konečně grafu vývoje chyby `gMedianErr`. Jako zdroj dat pro tento graf slouží pole dostupná přes metody třídy `VivaldiSimulation` s názvem `get*ErrData()`.

Při pozastavení a pokračování v simulaci (viz příloha A.2) dochází především k pozastavení, resp. spuštění simulačních vláken nastavením atributu `simulation` na `false`, resp. `true`. Současně jsou zastaveny, resp. znovu spuštěny časovače pro překreslení a sběr statistických údajů o chybě sítě. Na úrovni formuláře `VivaldiMonitorView` je zastaven, resp. spuštěn časovač `tmrRedrawNetwGraph`, zodpovědný za překreslování grafů.

5.5 VivaldiMonitor pro algoritmus Vivaldi s pevným časovým krokem

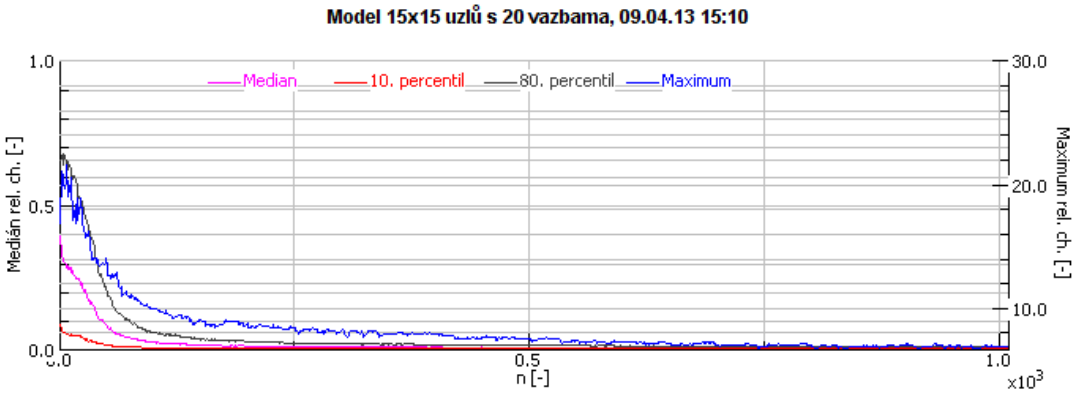
Tato varianta má téměř identické rozhraní a ovládání jako výše uvedená varianta simulující síť uzlů s adaptivním krokem. Jediným rozdílem jsou jiné parametry uzlů – pouze časový krok. Vše ostatní je naprosto shodné, takže je velmi snadné provést obdobnou simulaci i pro síť s konstantním krokem. Její význam je především v možnosti porovnání schopností predikce algoritmu Vivaldi s konstantním krokem ve srovnání s jeho variantou s krokem adaptivním.

6 PROVEDENÉ SIMULACE

6.1 Simulace uměle vytvořených sítí

V rámci semestrálního projektu byly provedeny některé simulace velmi jednoduchých umělých sítí. Složitější simulace by ani nebyly vhodné - knihovna, použitá ve vyvinuté aplikaci vyžadovala zejména odladit použitou implementaci algoritmu Vivaldi a její ovládání v programu. K tomuto účelu dobře posloužily simulace jednoho volného uzlu ukotveného dvěma stacionárními uzly a posléze jednoduché sítě 5 uzlů, tvořící čtverec. Zpočátku byly opět některé uzly ukotvené, po odladění knihovny i programu bylo možné provést úspěšnou simulaci zmíněné sítě 5 uzlů a následně i čtvercové mřížky 9 uzlů bez ukotvení [57].

Na tyto simulace navazuje diplomová práce Bc. Tomáše Handla [26], který provedl další simulace s většími umělými topologiemi, a to čtvercovými mřížkami uzlů o velikosti 15×15 a 20×20, tedy celkem s 225 a 400 uzly. Následující shrnutí se týká simulací pro adaptivní krok. Pro porovnání byly provedeny i simulace pro algoritmus s konstantním krokem, který zejména potvrdil tendenci uzlů mírně oscilovat kolem nejvhodnějšího umístění. Také potvrdil optimální velikost kroku δ cca 0,3, určenou v [57] pozorováním chování sítě. Typický průběh chyby pro 225 uzlů je zachycen na obrázku 6.1.



Obr. 6.1: Ukázka vývoje chyby sítě 225 uzlů algoritmu Vivaldi s konstantním krokem, $\delta = 0{,}3$. Převzato z [26].

Pro algoritmus s proměnným krokem byly postupně provedeny s cílem prozkoumat závislost rychlosti konvergence na parametrech algoritmu a počtu sousedů každého uzlu. Jako optimální hodnota konstanty c_c byla stanovena hodnota blízká jedné. Toto nastavení je však vhodné pouze na těchto umělých topologiích, v reálných sítích je za optimální variantu považována hodnota konstanty $c_c = 0{,}25$, jako vhodná

hodnota c_e se pak při simulacích ukázala hodnota 0,5. Konečně při zkoumání závislosti na počtu sousedů se jako vhodný ukázal vyšší počet sousedů - práce uvádí jako vhodnou hodnotu 20 sousedů, tedy největší zkoumaný počet. Tyto závěry byly uvedeny shodně pro obě velikosti sítě.

6.2 Chování algoritmu pro reálná data

6.2.1 Simulace sady Meridian

Předchozí simulace, provedené v rámci jiné diplomové práce, celkem přesvědčivě ukázaly schopnost nastavit parametry algoritmu Vivaldi tak, aby rychle konvergoval bez zbytečných oscilací. Klíčové je ale chování algoritmu pro hodnoty naměřené přímo v síti Internet, která je především vlivem směrování charakteristická určitou mírou porušování trojúhelníkové nerovnosti a zhoršuje tak schopnost predikce vzdálenosti metodami založenými na metrických souřadnicových prostorech.

Vstupní data pro simulace v této kapitole pochází z měření provedeného v období 5.–13. 5. 2004 pomocí měření metodou King[1]. Tato sada vznikla měřením mezi 2500 DNS servery webových stránek, přičemž byla věnována velká pozornost rozprostření jednotlivých měření tak, aby výsledky měření nebyly vlastním měřením ovlivněny [37]. Výsledky jsou dostupné na stránce [37] po zadání několika nepovinných údajů. Výsledky měření jsou tvořeny záznamy horní trojúhelníkové matice zpoždění[2]. Každý záznam pak obsahuje identifikátory uzlů, jejichž vzájemné zpoždění bylo měřeno a výsledné RTT udané v mikrosekundách, určené jako medián deseti měření.

Pro účely simulací byly z této sady měření vybrány náhodně podgrafy tří velikostí (60, 120 a 300 uzlů) s různým počtem sousedů (5, 10, 20 a 50 sousedů). Vlastní zpracování bylo provedeno vždy náhodným výběrem daného počtu uzlů z celkového počtu 2500 uzlů a uložením těchto uzlů a všech jejich spojů do objektu třídy `NetworkModel`, čímž se po jeho serializaci získal standardní soubor modelu pro `VivaldiMonitor`. Tyto soubory tak mohou sloužit jako reference pro další simulace, jelikož každý model tvoří úplný graf. Následně byl na takto vzniklé referenční modely aplikován program `NMVyberSousedy` (byl popsán v předchozí kapitole) pro získání částečných modelů s dříve zmíněnými počty sousedů.

[1] Metoda byla popsána v části 3.3.

[2] RTT je obousměrné zpoždění, takže měření v opačném směru dá obdobné výsledky, je-li provedeno správně.

6.2.2 Referenční simulace sady Meridian

Jako reference ke zhodnocení vlivů velikosti sítě, počtu sousedů a vlivu parametrů byla zvolena síť 120 uzlů, náhodně vybraných ze sady Meridian. Cílem bylo zjištění dosahovaných chyb predikce pro porovnání s ostatními simulacemi. Dalším účelem bylo prozkoumat charakter závislosti chyby predikce na parametru c_e, který je v článku popisujícím algoritmus Vivaldi zmíněn jako „dolaďovací“ [15]. Parametr c_c byl ponechán na doporučené hodnotě 0,25 [15], použité v existujících implementacích algoritmu Vivaldi s adaptivním ČK. Tuto hodnotu potvrdily i simulace umělých topologií v práci [26].

Simulace byly provedeny pro 6 různých hodnot parametru c_e v rozsahu $\langle 0; 1 \rangle$, což je obvyklý rozsah hodnot pro koeficienty. Simulace byly ukončeny vždy po asi 1000 iteracích[3] jednoho uzlu, dosažené relativní a absolutní chyby jsou shrnuty v tabulkách 6.1, resp. reftab:abs-120-plna. Pro vyhodnocení charakteru rozložení slouží sada percentilů vypočtených ze získané sady hodnot chyby predikce. Percentil, označený jako $p$tý je statistický údaj, značící, že p % hodnot z množiny je menší, než hodnota toho percentilu. Nejvýznamějším prercentilem je u problematiky predikce vzdálenosti 50. percentil, označovaný jako tzv. medián. Důvodem je použití mediánu jako převažujícího kritéria por hodnocení kvality predikce.

Tab. 6.1: Závislost relativní chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů

	Relativní chyba při parametru c_e rovném					
Percentil	0,05	0,20	0,40	0,60	0,80	0,99
10	0,022	0,022	0,024	0,021	0,020	0,022
20	0,046	0,047	0,047	0,044	0,043	0,046
30	0,075	0,074	0,074	0,070	0,070	0,072
40	0,109	0,105	0,106	0,100	0,101	0,101
50	0,147	0,145	0,144	0,136	0,140	0,137
60	0,196	0,197	0,197	0,188	0,188	0,186
70	0,274	0,275	0,275	0,257	0,262	0,256
80	0,395	0,392	0,402	0,384	0,382	0,374
90	0,703	0,636	0,770	0,668	0,674	0,653

Vzhledem k nedeterministickému charakteru algoritmu a mírným výkyvům v chybě predikce je obtížné vzhledem k poměrně malým rozdílům hodnot určit jednoznačný charakter závislosti, zdá se však, že všeobecně je nejmenší chyba dosahována pro

[3] Ukončení po 1000 iteracích platí i pro následující simulace.

Tab. 6.2: Závislost absolutní chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů

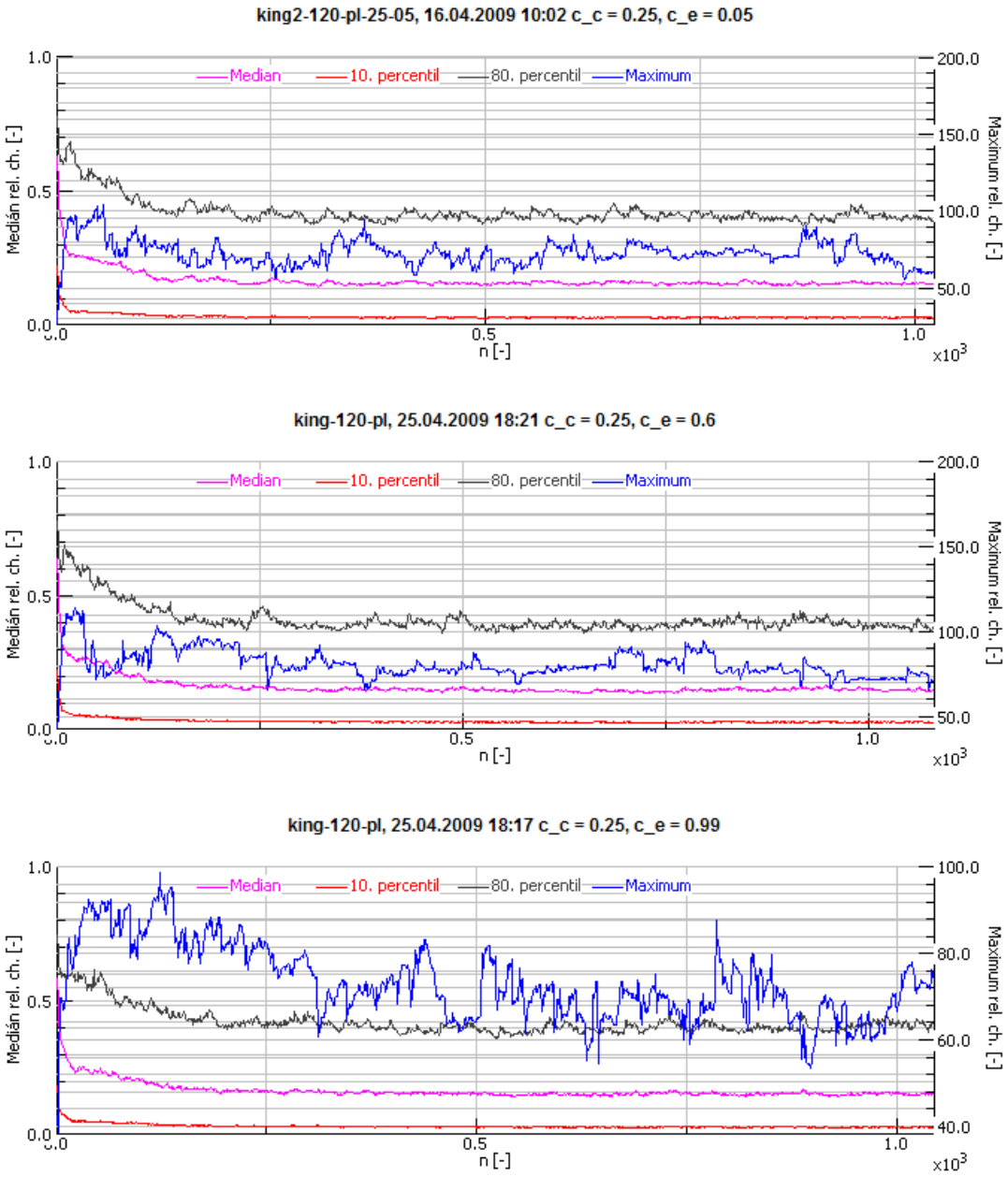
	Absolutní chyba při parametru c_e rovném					
Percentil	0,05	0,20	0,40	0,60	0,80	0,99
10	1,60	1,58	1,59	1,51	1,49	1,49
20	3,27	3,21	3,20	3,16	3,06	3,11
30	5,17	4,93	4,98	4,90	4,76	4,74
40	6,97	6,86	6,73	6,79	6,59	6,53
50	9,23	9,05	8,99	8,66	8,60	8,56
60	11,73	11,54	11,86	11,00	11,10	11,09
70	14,99	14,92	15,17	13,99	14,40	14,44
80	20,05	19,79	20,80	18,36	19,15	19,10
90	29,56	29,19	32,87	27,05	28,73	28,88

hodnoty parametru c_e v okolí hodnoty 0,5. Pravděpodobným vysvětlením je v případě hodnot c_e blízkým 1 vliv velkých výkyvů hodnoty chyb uzlů e_i, které způsobují mírné oscilace sítě, podobné chování simulovaných sítí v případě algoritmu s konstantním krokem. V případě malých hodnot hraje zřejmě roli počáteční nastavení vlastní chyby na hodnotu 100. To vede při velké váze předchozích hodnot vlastní chyby uzlu k pomalému zmenšování vlastní chyby, důsledkem je nedůvěra uzlů ve své souřadnice, což opět vede k oscilacím.

Příklady průběhu vývoje chyby predikce pro $c_e = 0{,}05$, 0,6 a 0,99 jsou zachyceny na obrázku 6.2.

Hodnoty kolem 0,5 představují jakýsi kompromis mezi oběma extrémy. Je však otázkou, jak bude závislost vypadat při jiných počátečních hodnotách chyby e_i. Zjištěná závislost odpovídá i simulacím z práce [36], která simulovala chování pro c_e rovné 1, 0,1 a 0,01. Například pro $c_e = 1$ se medián ustálil na chybě o asi 2 ms větší než pro $c_e = 0,1$.

Pro ověření byly kromě simulace plné sítě 120 uzlů simulována i síť 120 uzlů s 20 sousedy. Výsledky jsou opět zaznamenány do dvojice tabulek, tabulka 6.3 obsahuje hodnoty pro relativní chybu, tabulka 6.4 pak pro chybu relativní. Z hodnot je patrný podobný charakter závislosti, jako u plné sítě, a to nezávisle na tom, zda jde o závislost chyby absolutní nebo relativní, či na tom, zda jde o predikci spojů k sousedům či ne. Predikce RTT s uzlům, se kterými uzel ještě nekomunikoval, ale je součástí stejné sítě, se tedy odlišuje pouze mírně vyšší chybou. Odráží však kvalitu predikce k uzlům, vůči kterým určuje svou polohu, což znamená, že umístění vůči části uzlů sítě dostačuje k dobré predikci vzdálenosti k libovolnému uzlu.



Obr. 6.2: Vývoj chyby algoritmu Vivaldi, sada Meridian 120 uzlů, $c_c = 0{,}25$, $c_e = 0{,}05$; 0,6; 0,99

Tab. 6.3: Závislost relativní chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů s 20 sousedy

	Rel. chyba predikceznámých spojů [-]						Rel. chyba predikce neznámých spojů [-]					
	c_e [-]						c_e [-]					
Perc.	0,05	0,20	0,40	0,60	0,80	0,99	0,05	0,20	0,40	0,60	0,80	0,99
10	0,022	0,024	0,024	0,024	0,027	0,021	0,026	0,026	0,027	0,026	0,031	0,026
20	0,050	0,047	0,043	0,045	0,055	0,045	0,057	0,052	0,052	0,054	0,061	0,057
30	0,079	0,072	0,068	0,071	0,087	0,071	0,094	0,085	0,084	0,083	0,096	0,091
40	0,112	0,102	0,100	0,103	0,120	0,105	0,128	0,122	0,119	0,121	0,133	0,128
50	0,152	0,144	0,142	0,144	0,161	0,144	0,170	0,168	0,168	0,166	0,180	0,170
60	0,207	0,196	0,195	0,196	0,216	0,197	0,230	0,227	0,230	0,222	0,242	0,230
70	0,285	0,271	0,258	0,271	0,289	0,275	0,314	0,317	0,319	0,298	0,331	0,314
80	0,405	0,400	0,377	0,377	0,404	0,402	0,449	0,479	0,461	0,437	0,484	0,449
90	0,739	0,811	0,709	0,737	0,748	0,698	0,823	0,946	0,835	0,832	0,875	0,823

Z analýzy zanamenaných dat predikce spojů získaných simulací je také patrné, že charakter závislosti jednotlivých percentilů je velmi podobný, v následujících tabulkách je proto uváděn jen 10., 50. a 80. percentil, které jsou v průběhu simulace zaznamenávány aplikací VivaldiMonitor do grafu na kartě Data uzlů.

6.2.3 Simulace vzorku 60 uzlů sady Meridian

První simulovanou topologií byla síť 60 uzlů. Závislost hodnot jednotlivých vybraných percentilů na počtu známých sousedů každého uzlu jsou uvedeny v tabulkách 6.5 a 6.6. Ze získaných dat je na první pohled patrný výrazný vliv počtu sousedů uzlu. Pokud uzel neověřuje svoji pozici vůči alespoň 10 ostatním uzlům, predikce zpoždění se znatelně zhorší, zejména u predikce zpoždění k uzlům, které nejsou sousedy. V tomto případě jde o zvýšení u o přibližně 0,03 v případě relativní chyby predikce zpoždění k sousedům, resp. 3 ms v případě absolutní chyby. U neznámých spojů je rozdíl mezi chybou pro 5 uzlů a pro 10 (a více) dvojnásobek. Takový rozdíl by pravděpodobně značně snižoval relevanci predikovaných hodnot.

6.2.4 Simulace vzorku 120 uzlů sady Meridian

Průběhy pro síť 120 uzlů, opět náhodně vybraných, jsou uvedeny v tabulkách 6.7 a 6.8. Od počtu 10 sousedů (tj. 8,3 % spojů celkem) chyba predikce klesá se zvyšováním počtu sousedů. Neshoda mezi chybou známých a neznámých spojů se poněkud

Tab. 6.4: Závislost chyby algoritmu Vivaldi na c_e, sada Meridian, 120 uzlů s 20 sousedy

	Rel. chyba predikce známých spojů [-]						Rel. chyba predikce neznámých spojů [-]					
	c_e [-]						c_e [-]					
Perc.	0,05	0,20	0,40	0,60	0,80	0,99	0,05	0,20	0,40	0,60	0,80	0,99
10	1,76	1,51	1,65	1,603	1,96	1,57	1,90	1,79	1,93	1,85	2,13	1,93
20	3,48	3,18	3,14	3,22	3,63	3,31	3,88	3,62	3,83	3,74	4,13	3,81
30	5,31	4,93	4,75	4,98	5,52	5,01	6,00	5,77	5,88	5,88	6,29	5,77
40	7,12	7,15	6,69	6,99	7,49	7,14	8,08	8,08	8,01	8,02	8,71	7,87
50	9,26	9,61	8,70	9,18	10,02	9,19	10,96	10,62	10,56	10,38	11,40	10,34
60	12,27	12,24	11,192	11,53	12,68	11,61	14,02	14,05	13,52	13,02	14,60	13,52
70	15,49	15,90	14,31	14,68	16,37	14,99	17,83	18,44	17,03	16,75	18,69	17,28
80	20,67	20,90	19,19	19,67	21,59	19,67	23,05	25,02	22,53	22,30	25,11	22,70
90	31,36	32,17	28,50	28,04	32,72	28,60	34,76	37,44	32,68	33,63	37,93	33,70

Tab. 6.5: Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 60 uzlů

	Chyba predikce známých spojů [-]				Chyba predikce neznámých spojů [-]			
Perc.	5 vazeb	10 vazeb	20 vazeb	50 vazeb	5 vazeb	10 vazeb	20 vazeb	50 vazeb
10	0,024	0,021	0,024	0,022	0,053	0,022	0,024	0,023
50	0,158	0,128	0,126	0,126	0,273	0,142	0,152	0,133
80	0,408	0,329	0,324	0,312	0,651	0,357	0,400	0,330

snížila, ale stále je velmi výrazná. Chyba predikce se vzhledem k hodnotám pro síť 60 uzlů celkově zvýšila, v případě relativní chyby jde asi o 2 %.

6.2.5 Simulace vzorku 300 uzlů sady Meridian

Simulace sítí s 300 uzly dávají obdobné výsledky jako předchozí dva případy, viz tabulky 6.9 a 6.10. Chyba predikce se opět celkově zvýšila. Pro známé spoje jde asi o 2 % v případě relativní chyby, 0,7 ms v případě chyby absolutní.

U této sady bylo patrné zvýšení zátěže procesoru větším počtem paralelně běžících vláken simulujících algoritmus Vivaldi. Také bylo patrné zvýšení paměťových nároků vlivem ukládání záznamů jednotlivých iterací algoritmu do operační paměti. S narůstajícím počtem uzlů v simulované síti pak vlivem paměťových nároků dochází k prodlevám odezvy grafického rozhraní a systému. To bylo na obou počítačích použitých pro simulace (konfigurace viz tabulka 6.11) zde uvedených sítí patrné především při pokusu o simulaci plné matice 1000 uzlů (tj. pouze 20 % celé

Tab. 6.6: Závislost chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 60 uzlů

	Chyba predikce známých spojů [ms]				Chyba predikce neznámých spojů [ms]			
Perc.	5 vazeb	10 vazeb	20 vazeb	50 vazeb	5 vazeb	10 vazeb	20 vazeb	50 vazeb
10	1,69	1,48	1,55	1,59	4,19	1,65	1,79	1,79
50	11,31	8,26	9,07	8,72	19,37	9,41	10,54	9,26
80	23,21	19,50	20,54	19,04	39,19	21,58	23,62	20,43

Tab. 6.7: Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 120 uzlů

	Chyba predikce známých spojů [-]				Chyba predikce neznámých spojů [-]			
Perc.	5 vazeb	10 vazeb	20 vazeb	50 vazeb	5 vazeb	10 vazeb	20 vazeb	50 vazeb
10	0,020	0,025	0,025	0,022	0,040	0,030	0,025	0,024
50	0,139	0,144	0,150	0,148	0,264	0,196	0,166	0,153
80	0,404	0,441	0,384	0,403	0,679	0,512	0,460	0,392

Tab. 6.8: Závislost chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 120 uzlů

	Chyba predikce známých spojů [ms]				Chyba predikce neznámých spojů [ms]			
Perc.	5 vazeb	10 vazeb	20 vazeb	50 vazeb	5 vazeb	10 vazeb	20 vazeb	50 vazeb
10	1,54	1,78	1,68	1,55	2,84	2,16	1,82	1,64
50	8,72	9,62	9,44	9,16	16,59	12,21	10,40	9,43
80	20,07	20,34	20,39	19,94	34,77	26,58	23,85	20,39

sady Meridian), vytvořené opět náhodným výběrem. Simulaci bylo možné spustit, ale probíhala velmi pomalu a velmi zatěžovala systém. Hlavním důvodem je zřejmě režie přepínání mezi větším počtem vláken, další problémem jso značné nároky na paměť, způsobené ukládáním záznamů historie uzlů do paměti.

Tab. 6.9: Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 300 uzlů

	Chyba predikce známých spojů [-]					Chyba predikce neznámých spojů [-]				
Perc.	5 vazeb	10 v.	20 v.	50 v.	100 v.	5 vazeb	10 v.	20 v.	50 v.	100 v.
10	0,028	0,026	0,028	0,029	0,026	0,057	0,038	0,031	0,029	0,027
50	0,216	0,181	0,176	0,173	0,169	0,348	0,235	0,196	0,182	0,173
80	0,567	0,467	0,441	0,458	0,449	0,921	0,598	0,500	0,488	0,459

Tab. 6.10: Závislost chyby predikce algoritmu Vivaldi na počtu sousedů, sada Meridian, 300 uzlů

	Chyba predikce známých spojů [ms]					Chyba predikce neznámých spojů [ms]				
Perc.	5 vazeb	10 v.	20 v.	50 v.	100 v.	5 vazeb	10 v.	20 v.	50 v.	100 v.
10	1,91	1,69	1,88	1,77	1,73	3,79	2,61	2,03	1,89	1,77
50	12,42	10,62	10,30	10,42	9,86	21,53	14,16	11,43	10,82	10,16
80	28,64	22,57	22,01	22,48	21,56	45,80	29,43	24,38	23,77	21,98

Pro rozsáhlejší simulace je tedy třeba provést úpravy jak v použité knihovně, tak ve vlastním ovládacím programu. Zejména jde o odkládání veškerých záznamů (`NodeHistorieRecord`) do dočasných souborů na disk, nezávisle na odkládání nepoužívaných bloků paměti na disk, prováděného automaticky operačním systémem. Rovněž je možné po spuštění simulace odstranit model sítě z paměti a načíst ho zpět jen v případě potřeby, např. při restartu simulace.

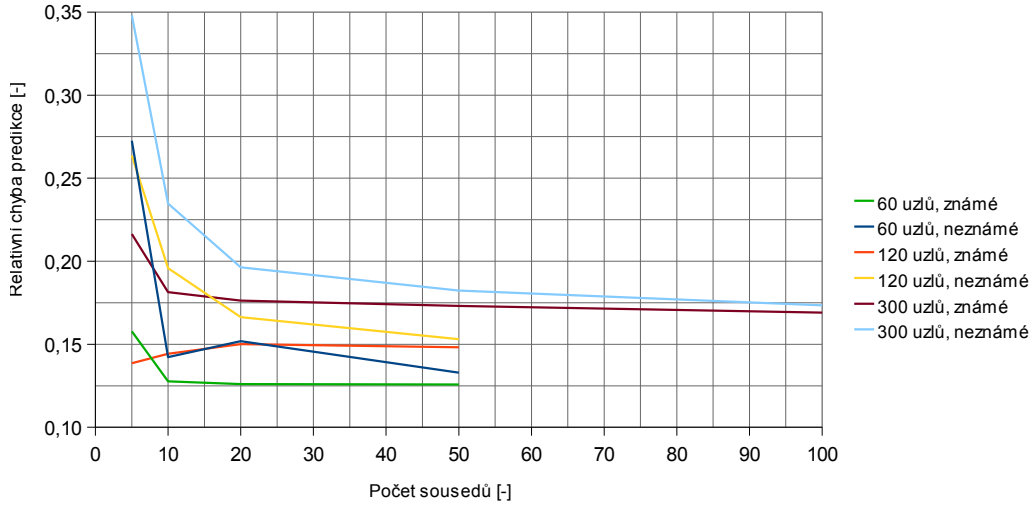
Tab. 6.11: Konfigurace použité pro simulace sady Meridian

Parametr	Konfigurace 1	Konfigurace 2
Procesor	Pentium Core 2 Duo 6300	Pentium Core 2 Duo T7500
Pracovní kmitočet	1,86 GHz	2,2 GHz
Kmitočet systémové sběrnice	200 MHz	400 MHz
Operační paměť	2 × 500 MiB DDR2	3 GiB DDR3

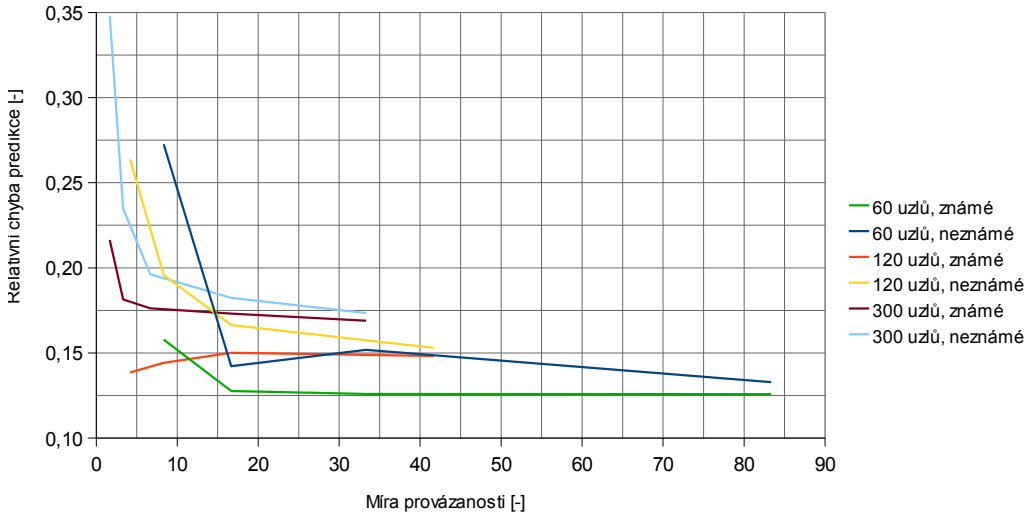
Těmito kroky by byla vyřešena paměťová náročnost simulací. Stále by však přetrval problém velké režie způsobené přepínáním mezi stovkami vláken. Spuštění samostatného vlákna (procesu) pro každou simulovanou entitu a s tím spojený nízký

výkon je typický pro tzv. simulace založené na procesech [39] (*process-based simulation*). Pro rozsáhlé simulace se proto používají simulace událostní (*event-based simulation*, *discrete event simulation*), které jsou základem moderních simulátorů, jako je např. *ns2* [2] nebo *p2psim* [32].

6.2.6 Shrnutí simulací sady Meridian



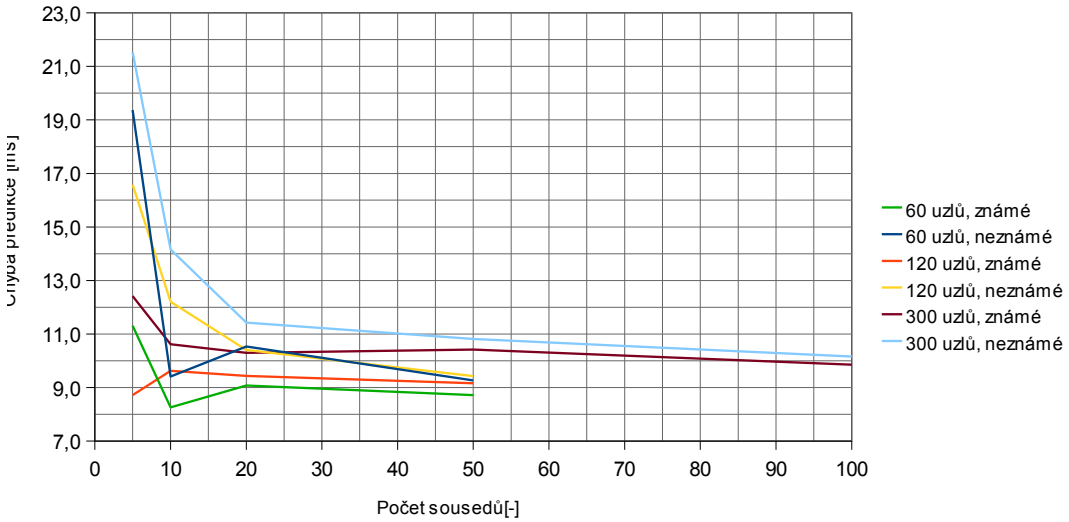
Obr. 6.3: Závislost relativní chyby predikce algoritmu Vivaldi na počtu sousedů a velikosti sítě, sada Meridian



Obr. 6.4: Závislost relativní chyby predikce algoritmu Vivaldi na provázanosti a velikosti sítě, sada Meridian

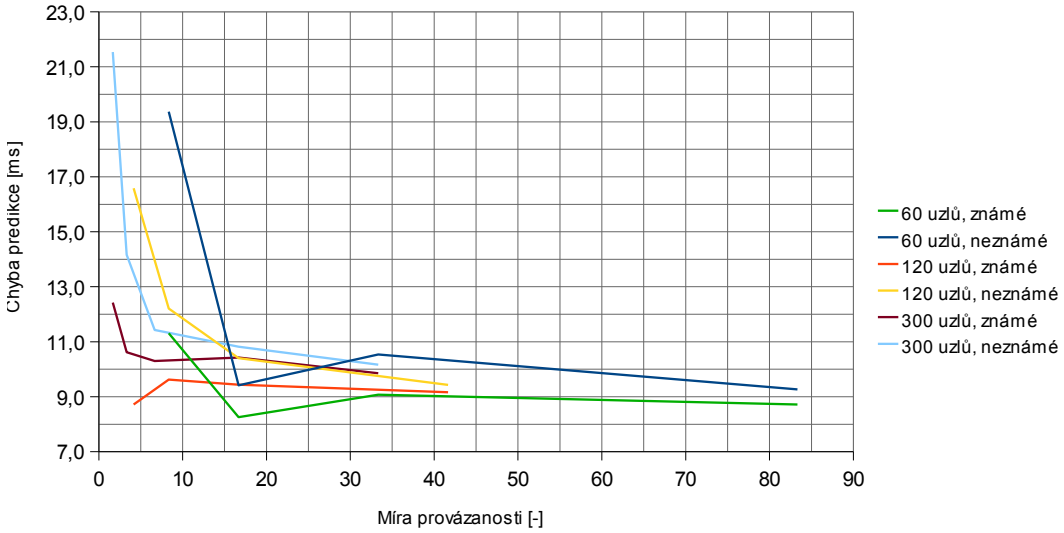
Z provedených simulací vyplývá několik zřejmých charakteristik predikce zpoždění pomocí algoritmu Vivaldi. Predikce je spolehlivá v širokém rozmezí parametrů, medián relativní chyby provedených simulací je typicky 14–18 %, medián absolutní chyby 9–11 ms, viz grafy na obr. 6.3, 6.2.6 Rychleji k menším chybám konvergují menší sítě, ale je nutné, aby každý uzel určoval svoji pozici vzhledem k dostatečnému počtu dalších uzlů, v opačném případě je predikce zatížena velkou chybou. Pro dostatečnou shodu nicméně postačuje poměrně malý podíl počtu sousedů k celkovému počtu uzlů (dále míra provázanosti) – viz obr. 6.6, 6.2.6.

Velmi zajímavý je pokles míry provázanosti potřebné k dosažení stejné míry shody chyby predikce zpoždění pro známé a neznámé spoje. Jedno z možných vysvětlení spočívá ve větší různorodosti zpoždění mezi stanicemi. Oproti plné sadě Meridian se v částečných sadách nachází méně spojů. Ačkoli náhodným výběrem z celé sady bude charakter histogramu hodnot zpoždění zachován, získáme množinu s dosti omezeným výběrem. To lze částečně přirovnat k situaci vyvolané výběrem jen nejbližších sousedů uzlu, vedoucím ke špatné predikci vzdáleností k vzdálenějším uzlům [15].



Obr. 6.5: Závislost chyby predikce algoritmu Vivaldi na počtu sousedů a velikosti sítě, sada Meridian

Zjištěné absolutní chyby jsou srovnatelné (s přihlédnutím k rozsahu simulované topologie) s výsledky v práci [36]. Zde dosahovala chyba pro parametr $c_c = 0{,}25$, (hodnota parametru c_e nebyla uvedena) přibližně 17 % pro simulaci úplné sady King1.



Obr. 6.6: Závislost chyby predikce Vivaldi na provázanosti a velikosti sítě, sada Meridian

6.2.7 Simulace sady King1

Nezávisle na simulacích sady Meridian byly v rámci diplomové práce Bc. Tomáše Handla rovněž provedeny simulace na vzorku dat naměřených v reálné síti. Pro simulaci byla vybrána sada King1. Z této sady bylo po vyloučení měření s RTT mimo rozsah 1 – 1000 ms vybrána množina 80 uzlů s 10, 20 a 50 náhodně vybranými sousedy. Sledována pak byla závislost mediánu chyby na počtu sousedů a velikosti konstanty c_c, omezujícího shora velikost adaptivního kroku. Podle očekávání větší hodnoty konstanty c_c vede k větší výsledné chybě predikce. Důvodem jsou opět zvýšené oscilace, které zabrání umístění uzlů, čímž je celá síť rozkmitávána, podobně jako u algoritmu s konstantním krokem. Chyba pro doporučovanou hodnotu $c_c = 0{,}25$ dosahovala pro 20 vazeb asi 9,5 %. Stejně jako u simulací části sady Meridian byla chyba nepřijatelně vysoká, pokud počet sousedů poklesl pod 10.

7 ZÁVĚR

V této práci byly prozkoumány hlavní představitelé systémů odhadu pozice a vzdáleností v sítí Internet a byly krátce zhodnoceny jejich vlastnosti, výhody a nevýhody. Vzhledem k neexistenci standardů ani spuštěných centralizovaných služeb byl k dalšímu studiu zvolen distribuovaný, decentralizovaný algoritmus Vivaldi, využívající umělé souřadnicové systémy k odhadům zpoždění mezi uzly – stanicemi, servery apod. – v síti Internet. Tento algoritmus je zajímavý zejména svým distribuovaným charakterem, který je vhodný pro moderní překryvné sítě.

Stěžejním výsledkem práce je vyvinutá simulační aplikace VivaldiMonitor. Tato aplikace umožňuje simulovat sítě až několika stovek uzlů implementujících algoritmus Vivaldi s adaptivním ČK. Aplikace slouží k sestavení (či načtení) modelu sítě, který má být simulován, k ovládání a vyhodnocení simulace. Nedílnou součástí je i možnost uložení dat simulace pro další zpracování. Aplikace umožňuje např. uložit průběhy proměnných každého z uzlů a průběh celkové chyby sítě ve formátu vhodném pro zpracování pomocí prostředí Matlab. Dále je možné posoudit přesnost predikce zpoždění a tabulku hodnocených spojů a jejich chyb predikce uložit jako datový soubor, kompatibilní s tabulkovými procesory jako např. Excel nebo Calc. Použitá simulační knihovna vivaldi-lib, vyvinutá v rámci diplomové práce Bc. Tomáše Handla [26] umožňuje simulace sítí do velikosti několika stovek uzlů.

Simulace, navazující na jednoduché simulace umělých sítí z práce [57], potvrdily schopnost rychlé konvergence pomocí algoritmu Vivaldi pro rozsáhlejší umělé sítě. Schopnost dosáhnout uspokojivé absolutní i relativní chyby predikce byla následně potvrzena na modelech vycházejících z měření zpoždění vybraných uzlů v síti Internet. V rámci této práce byla sledována závislost chyby predikce na počtu sousedů každého uzlu. Simulace byly provedeny na části sady měření Meridian. Obě simulace potvrdily schopnost predikovat zpoždění s chybou asi 10–13 ms bez výrazné závislosti na použitých konstantách, je-li dodržen minimální počet (více než 10) sousedů.

LITERATURA

[1] ABRAHAO, B. – KLEINBERG, R. *On the Internet Delay Space Dimensionality* [online]. In *ACM/SIGCOMM Internet Measurement Conference (IMC'08), Vouliagmeni, Greece.* [s.l.]: [s.n.], 2009. s. 157 – 168. URL: <http://www.cs.cornell.edu/~abrahao/docs/imc08.pdf> [cit. 25. 4. 2009]

[2] AMIR, E. – FALL, K. – McCANNE, S. et al. *The Network Simulator - ns-2* [online]. [s.l.]: Information Sciences Institute, 2006. URL: <http://www.isi.edu/nsnam/ns/> [cit. 17. 5. 2009]

[3] azureuswiki.com. *Distributed hash table* [online]. URL: <http://azureuswiki.com/index.php/Distributed_hash_table> [cit. 18. 4. 2009].

[4] BAŠTINEC, J. – NOVÁK, M. *Moderní numerické metody* [online]. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2009. URL: <http://www.umat.feec.vutbr.cz/~novakm/MMNM2009.pdf> [cit. 2. 5. 2009].

[5] BEN AZZOUNA, N. – GUILLEMIN, F. *Experimental analysis of the impact of peer-to-peer applications on traffic in commercial IP networks* [online]. Lannion: Division R&D. France Telecom, 2004. URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.1857&rep=rep1&type=pdf> [cit. 10. 5. 2009].

[6] Blizzard Entertainment. *World of Warcraft Community site* [online]. URL: <http://www.worldofwarcraft.com> [cit. 16. 5 2009]

[7] BROOKSHAW, L. *Java 2D Graph Package* [online]. [s.l.]: University of Southern Queensland. Faculty of Sciences. Mathematics and Computing, 1996. URL: <http://www.sci.usq.edu.au/staff/leighb/graph> [cit. 16. 5. 2009]

[8] BerliOS. *JMathTools* [online]. URL: <http://jmathtools.berlios.de/doku.php> [cit. 17. 5. 2009].

[9] DEERING, S. – CHERITON, D. , *Multicast Routing in Datagram Networks and Extended LANS*, In *ACM Transactions On Computer Systems, Vol. 18, No. 2.* [s.l]: [s.n], květen 1990, s. 85–110. URL: <http://www-cse.ucsd.edu/classes/wi01/cse222/papers/deering-multicast-tocs90.pdf> [cit. 12. 4. 2009]

[10] CASTRO, M. et al. *PIC: Practical Internet Coordinates for Distance Estimation.* In *International Conference on Distributed Systems.* [s.l]: [s.n], 2003.

s. 178–187. URL: <http://www.cl.cam.ac.uk/Teaching/2003/AdvSysTop/pic.pdf> [cit. 23. 4. 2009].

[11] CASTRO, M. et al. *Secure routing for structured peer-to-peer overlay networks.* In *Proceedings of the 5th Usenix Symposium on Operating Systems Design and Implementation* (OSDI 2002). Boston: [s.n], prosinec 2002. URL: <http://www.cs.rice.edu/~dwallach/pub/osdi2002.pdf> [cit. 12. 4. 2009].

[12] CHEN,Y. – OVERTON, C. – KATZ, R., *Internet Iso-bar: A Scalable Overlay Distance Monitoring System* [online]. in *Journal of Computer Resource Management, Computer Measurement Group. Spring Edition.* [s.l.]: [s.n.], 2002. URL: <http://www.cs.northwestern.edu/~ychen/research/wnmms/isobar.ps> [cit. 13. 4. 2009].

[13] COX, R. – DABEK, F. *Learning Euclidean coordinates for Internet hosts* [online]. Cambridge (USA) : Massachusetts Institute of Technology, 2002. URL: <http://www.txtr.com/text/ak489> [cit. 6. 12. 2008].

[14] COX, R. – DABEK, F. *Practical, Distributed Network Coordinates.* In *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)* [online], Cambridge (USA): Massachusetts Institute of Technology, 2003. URL: <http://pdos.csail.mit.edu/papers/hotnets:vivaldi/paper.pdf> [cit. 3. 12. 2008].

[15] DABEK, F. et al. *Vivaldi: A Decentralized Network Coordinate System.* In *Proceedings of SIGCOMM 2004* [online], Portland: Massachusetts Institute of Technology, 2004. URL: <http://pdos.csail.mit.edu/papers/vivaldi:sigcomm/paper.pdf> [cit. 5. 12. 2008].

[16] DOVAL, D. – O'MAHONY, D. *Overlay networks: A scalable alternative for P2P* [online]. *IEEE Internet computing.* [s.l]: [s.n], červenec – srpen 2003. URL: <http://www.dynamicobjects.com/papers/w4spot.pdf> [cit. 15. 12. 2008].

[17] eMule Project. *Oficiální stránky eMule* [online]. URL: <http://http://www.emule-project.net/home/perl/general.cgi?l=32> [cit. 10. 5. 2009].

[18] FOWLER, A.: *A Swing Architecture Overview: The Inside Story on JFC Component Design* [online]. URL: <http://http://java.sun.com/products/jfc/tsc/articles/architecture> [cit. 18. 4. 2009].

[19] FRANCIS, P. *IDMaps: A Global Internet Host Distance Estimation Service.* In *IEEE/ACM Trans. on Networking* [online], New York: [s.n], říjen

2001. URL: <http://http://idmaps.eecs.umich.edu/papers/ton01.pdf> [cit. 7. 12. 2008].

[20] FRITSCH, T. – RITTER, H. – SCHILLER, J. *The Effect of Latency and Network Limitations on MMORPGs.* In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games* [online], Hawthorne: ACM, 2005. URL: <http://www.research.ibm.com/netgames2005/papers/fritsch.pdf> [cit. 15. 12. 2008].

[21] The Gnutella Developer Forum. *The Annotated Gnutella Protocol Specification v0.4.* [online]. URL: <http://rfc-gnutella.sourceforge.net/developer/stable/index.html> [cit. 14. 12. 2008].

[22] GUMMADI, K. – SAROIU, S. – GRIBBLE, S. *King: Estimating Latency between Arbitrary Internet End Hosts* [online]. In SIGCOMM Internet Measurement Workshop 2002. Seattle: University of Washington. Department of Computer Science & Engineering, 2002 URL: <http://www.mpi-sws.org/~gummadi/king/king.pdf> [cit. 12. 4. 2009].

[23] GWERTZMAN, J. *Autonomous Replication in Wide-Area Internetworks* [online]. Cambridge (USA): Harvard College, 1995. URL: <ftp://steward.harvard.edu/users/gwertzman/thesis.ps.gz> [cit. 15. 12. 2008].

[24] HALLIDAY, D. – RESNICK, R. – WALKER, J. *Fyzika.* Brno: VUTIUM, 2003. Dotisk 1. českého vydání. ISBN 80-214-1868-0. Část 1, kapitola 7, s. 153.

[25] HANDL, T. *Algoritmus Vivaldi pro nalezení stanic v internetu.* Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2008. Semestrální projekt. Vedoucí práce Ing. Dan Komosný, Ph.D.

[26] HANDL, T. *Algoritmus Vivaldi pro nalezení stanic v internetu.* Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2009. Diplomová práce. Vedoucí práce Ing. Dan Komosný, Ph.D.

[27] HAROLD, E. *Java introduction* [online]. URL: <http://www.cafeaulait.org/course/week1/index.html> [cit. 6. 12. 2008].

[28] Internet Systems Consortium. *Internet host count history* [online]. URL: <http://www.isc.org/solutions/survey/history> [cit. 14. 12. 2008]

[29] JAROŠOVÁ, L. *Měření vzdáleností stanic prostřednictvím ICMP protokolu v IP sítích.* Brno: Vysoké učení technické v Brně, Fakulta Elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2001, počet stran 63. Bakalářská práce. Vedoucí práce Ing. Radim Burget.

[30] Java.net. *An Introduction to the Swing Application Framework API (JSR-296)* [online].
URL: <`http://appframework.dev.java.net/intro/index.html`> [cit. 18. 4. 2009].

[31] KAASHOEK, M et al. *The „King" data set* [online]. Cambridge (USA): Massachusetts Institute of Technology. Computer Science and Artificial Intelligence Laboratory, 2003. URL: <`http://http://pdos.csail.mit.edu/p2psim/kingdata/`> [cit. 12. 4. 2009].

[32] KAASHOEK, M et al. *p2psim: A simulator for peer-to-peer (p2p) protocols.* Cambridge (USA): Massachusetts Institute of Technology. Computer Science and Artificial Intelligence Laboratory, 2004. URL: <`http://pdos.csail.mit.edu/p2psim/`> [cit. 12. 4 2009]

[33] KACZMARCZYK, G. *Downhill Simplex Method for Many (20) Dimensions* [online]. URL: <`http://paula.univ.gda.pl/~dokgrk/simplex.html`> [cit. 16. 12. 2008].

[34] KÁLLAY, F. – PENIAK, P. *Počítačové sítě a jejich aplikace: LAN/MAN/WAN.* Druhé, aktualizované vydání. Praha: Grada Publishing, 2003. 356 s.

[35] LEDLIE, J. – GARDNER, P. – SELTZER, M. *Network Coordinates in the Wild* [online]. In *In Proceedings of NSDI 2007.* Cambridge (USA): Harvard School of Engineering and Applied Sciences and Aelitis, 2007. URL: <`http://www.eecs.harvard.edu/~syrah/nc/wild07.pdf`> [cit. 25. 4 2009].

[36] LI-WEI, L. *PCoord: a decentralized network coordinate system for Internet distance prediction* [online]. Cambridge (USA): Massachusetts Institute of Technology, Dept. of Civil and Environmental Engineering, 2005. URL: <`http://dspace.mit.edu/bitstream/handle/1721.1/31134/61184096.pdf?sequence=1`> [cit. 12. 4. 2009].

[37] Meridian Project. *Meridian Data Description* [online]. URL: <`http://www.cs.cornell.edu/people/egs/meridian/data.php`> [cit. 19. 4. 2009].

[38] NAKAO, A. – PETERSON, L. – BAVIER, A. *A Routing Underlay for Overlay Networks* [online]. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, str. 11–19. New York: ACM, 2003. URL: <http://www.cs.princeton.edu/courses/archive/fall06/cos561/papers/nakao03.pdf> [cit. 15. 12. 2008].

[39] NAYEBI, A – MERAJI, S. – SHAMAEI, A. *Xmulator: A listener based, multi-layered simulator* [online]. [s.l.]: Sharif University of technology, 2006. URL: <http://www.xmulator.org/files/v1.0/doc/doc_en.pdf> [cit. 20. 4. 2009]

[40] NG, E. – HUI, Z. *Predicting Internet network distance with coordinates-based approaches* [online]. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies.*. [s.l.]: IEEE, 2002. str. 170-179, svazek 1. URL: <http://www.cs.rice.edu/~eugeneng/papers/INFOCOM02.pdf> [cit. 6. 12. 2008].

[41] ODLYZKO, A. *Growth of the Internet* [online]. Minneapolis: University of Minnesota, 2003?. URL: <http://www.dtc.umn.edu/~odlyzko/doc/itcom.internet.growth.pdf> [cit. 14. 12. 2008].

[42] PIAS, M. et al. *Lighthouses for scalable distributed location* [online]. URL: <http://research.microsoft.com/~tharris/papers/2003-iptps.pdf> [cit. 7. 12. 2008].

[43] PIETZUCH, P. et. al. *Network-Aware Overlays with Network Coordinates* [online]. In *Proceedings of the 26th IEEE International ConferenceWorkshops on Distributed Computing Systems* Cambridge (USA). Harvard University, 2006. URL: <http://www.eecs.harvard.edu/~syrah/nc/worlds05/worlds05-talk.pdf> [cit. 24. 4. 2009].

[44] PIETZUCH, P. – LEDLIE, J. – PARKER, M. *Pyxida: An Open Source Network Coordinate Library and Application* [online].
URL: <http://pyxida.sourceforge.net> [cit. 10. 5. 2009]

[45] PIETZUCH, P. – LEDLIE, J. – SELTZER, M. *Supporting Network Coordinates on PlanetLab* [online]. In *Proceedings of the 2nd conference on Real, Large Distributed Systems - Volume 2*, str. 19-24. URL: <http://www.eecs.harvard.edu/~syrah/nc/worlds05/worlds05-talk.pdf> [cit. 15. 12. 2008].

[46] PlanetLab Consortium. *PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services.*
URL: <http://www.planet-lab.org> [cit. 16. 5. 2009].

[47] POSTEL, J., *RFC793 - Transmission Control Protocol* [online]. [s.l.]: USC/Information Sciences Institute, září 1981.
URL: <http://www.faqs.org/rfcs/rfc793.html> [cit. 24. 4 2009].

[48] RAQAZZ, S.: *Design Pattern: Model-View-Presenter (MVP)* [online].
URL: <http://www.razzaq.org/pub/design_pattern_model_view_presenter.pdf> [cit. 9. 12. 2008]

[49] RATASAMY, S. et al. *CA Scalable Content-Addressable Network* [online]. In *Proc. of ACM SIGCOMM'01*, San Diego: [s.n], Aug. 2001. URL: <http://www.cs.cornell.edu/People/francis/p13-ratnasamy.pdf> [cit. 24. 4. 2009].

[50] RHEA, S. et al. *The Bamboo Distributed Hash Table: A Robust, Open-Source DHT* [online]. URL: <http://bamboo-dht.org> [cit. cit. 10. 5. 2009].

[51] SHALUNOV, S. et al. *A One-way Active Measurement Protocol (OWAMP).* [s.l.]: The Internet Society, 2006.
URL: <http://www.rfc-editor.org/rfc/rfc4656.txt> [cit. 16. 5 2009].

[52] SPENCE, D. *An implementation of a Coordinate based Location System* [online] Cambrigde (UK): University of Cambridge, 2003. URL: <http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-576.pdf> [cit. 16. 12. 2008].

[53] STEINER, M; – BIERSACK, E.: *Where is my peer? Evaluation of the Vivaldi Network Coordinate System in Azureus.* In *Networking 2009.* Aachen: 8th IFIP International Conference on Networking, květen 2009. URL: <http://www.eurecom.fr/~btroup/BPublished/StBi09whereismypeer.pdf> [cit. 11. 4. 2009].

[54] STOICA, I. et al. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications* [online]. In *Proc. of ACM SIGCOMM'01*, San Diego: [s.n], Aug. 2001. URL: <http://pdos.csail.mit.edu/papers/chord:sigcomm01> [cit. 24. 4. 2009].

[55] STRIBLING, J. *All-Pairs-Pings for PlanetLab* [online].
URL: <http://pdos.csail.mit.edu/~strib/pl_app/> [cit. 20. 4. 2008].

[56] Sun Microsystems: *Swing: Java Foundation Classes* [online]. URL: <http://java.sun.com/javase/6/docs/technotes/guides/swing/> [cit. 18. 4. 2009]

[57] ŠVÉDA, J. *Nalezení pozice stanic v Internetu pomocí uměle vytvořených souřadnicových systémů.* Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2008. 40 s. Semestrální práce. Vedoucí práce Ing. Dan Komosný, Ph.D.

[58] Telegeography. *Global Internet Map* [online]. URL: <http://www.telegeography.com/products/map_internet/index.php> [cit. 10. 12. 2008]

[59] Theory.org Wiki. *Bittorrent Protocol Specification v1.0* [online]. URL: <http://wiki.theory.org/BitTorrentSpecification> [cit. 14. 12. 2008].

[60] Vuze. Azureus – now called Vuze – BitTorrent client [online]. URL: <http://http://azureus.sourceforge.net> [cit. 20. 4. 2009].

[61] Wang, R. *Median Filter*. [s.l]: [s.n], 2004. URL: <http://fourier.eng.hmc.edu/e161/lectures/smooth_sharpen/node3.html> [cit. 16. 5. 2009].

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ADSL	asynchronous digital subscriber line
CU	cílový uzel
DHT	distributed hash table – distribuovaná hashovací tabulka
DNS	domain name system
GNP	global network positioning
GPS	global positioning system
HOPS	host proximity service
ICMP	internet control message protocol – základní signalizační protokol sady TCP/IP
IDMaps	internet distance map service
OB	orientační bod
PNG	portable network graphics
RTT	round trip time delay
TCP/IP	internet protocol – Transmission control protocol
$\mathbf{L}$	matice zpoždění mezi síťovými uzly
L_{ij}	zpoždění mezi dvěma síťovými uzly
L_j	zpoždění k vzdálenému síťovému uzlu
$\|\mathbf{x}\|$	velikost vektoru $\mathbf{x}$
u($\mathbf{x}$)	normalizovaný vektor $\mathbf{x}$, tedy jednotkový vektor souhlasně rovnoběžný s vektorem $\mathbf{x}$
δ	časový krok
w, w_i, w_{ij}	váhovací konstanta

SEZNAM PŘÍLOH

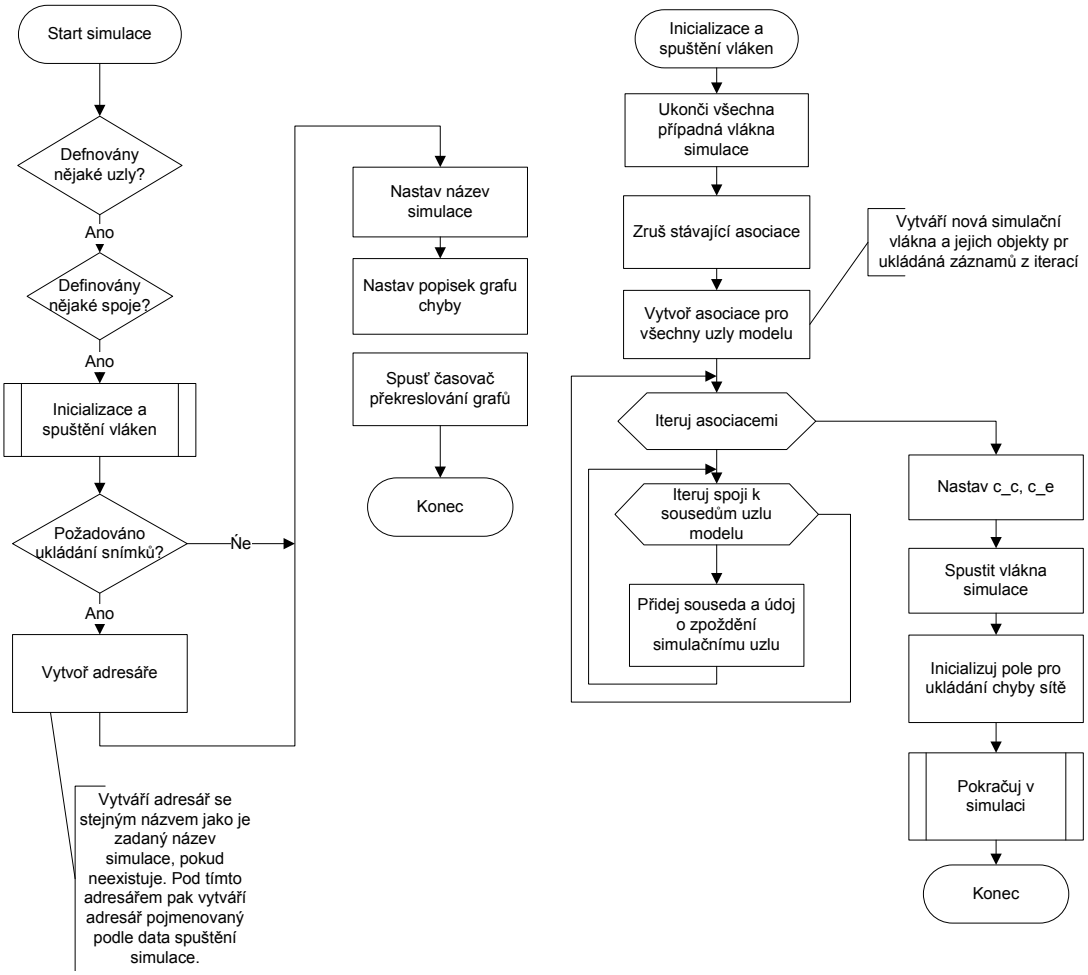
A Dokumentace simulačního programu VivaldiMonitor **78**
A.1 Spuštění simulace 78
A.2 Pozastavení a pokračování simulace 79
A.3 Diagram hlavních tříd aplikace 80
A.4 Diagram tříd modelu sítě 81
A.5 Vazby s knihovnou vivaldi-lib 82
A.6 Životní cyklus objektu Vivaldi_node2 83

B Přehledový diagram knihovny vivaldi-lib **84**

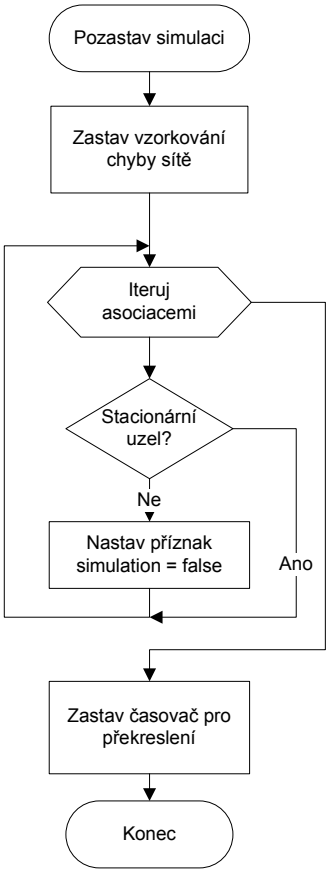
C Obsah přiloženého CD **85**

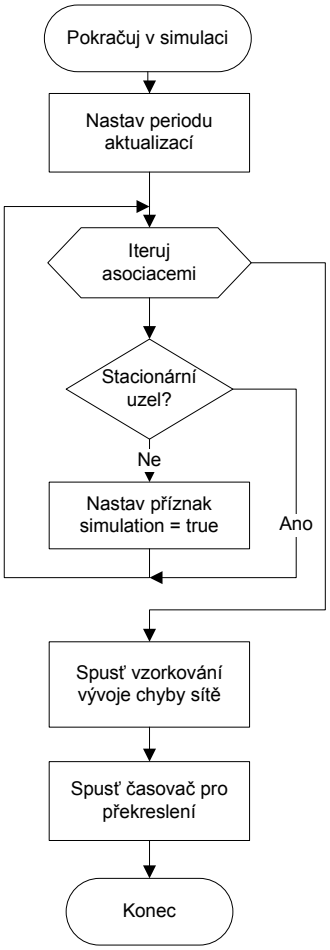
A DOKUMENTACE SIMULAČNÍHO PROGRAMU VIVALDIMONITOR

A.1 Spuštění simulace

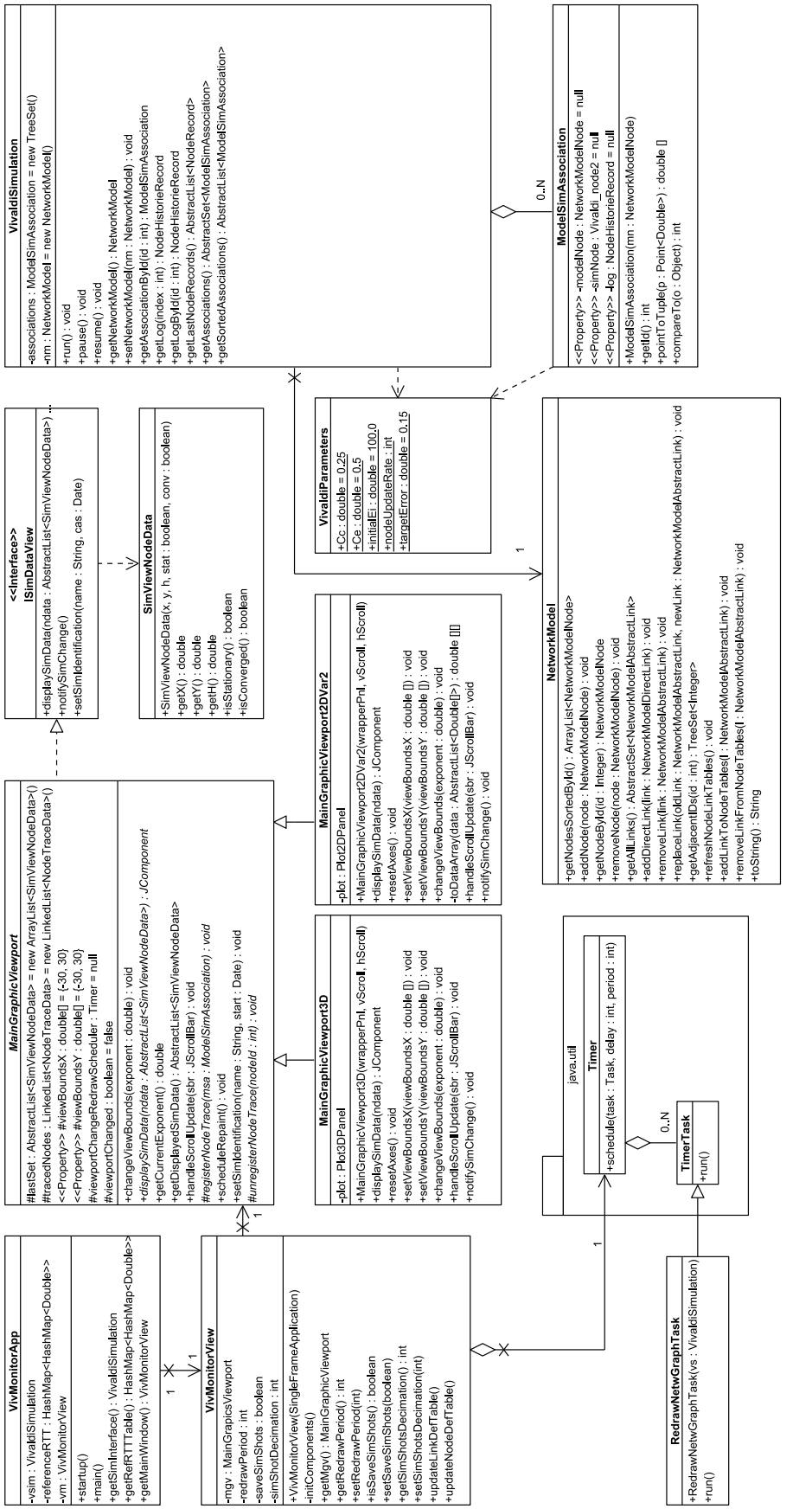


A.2 Pozastavení a pokračování simulace

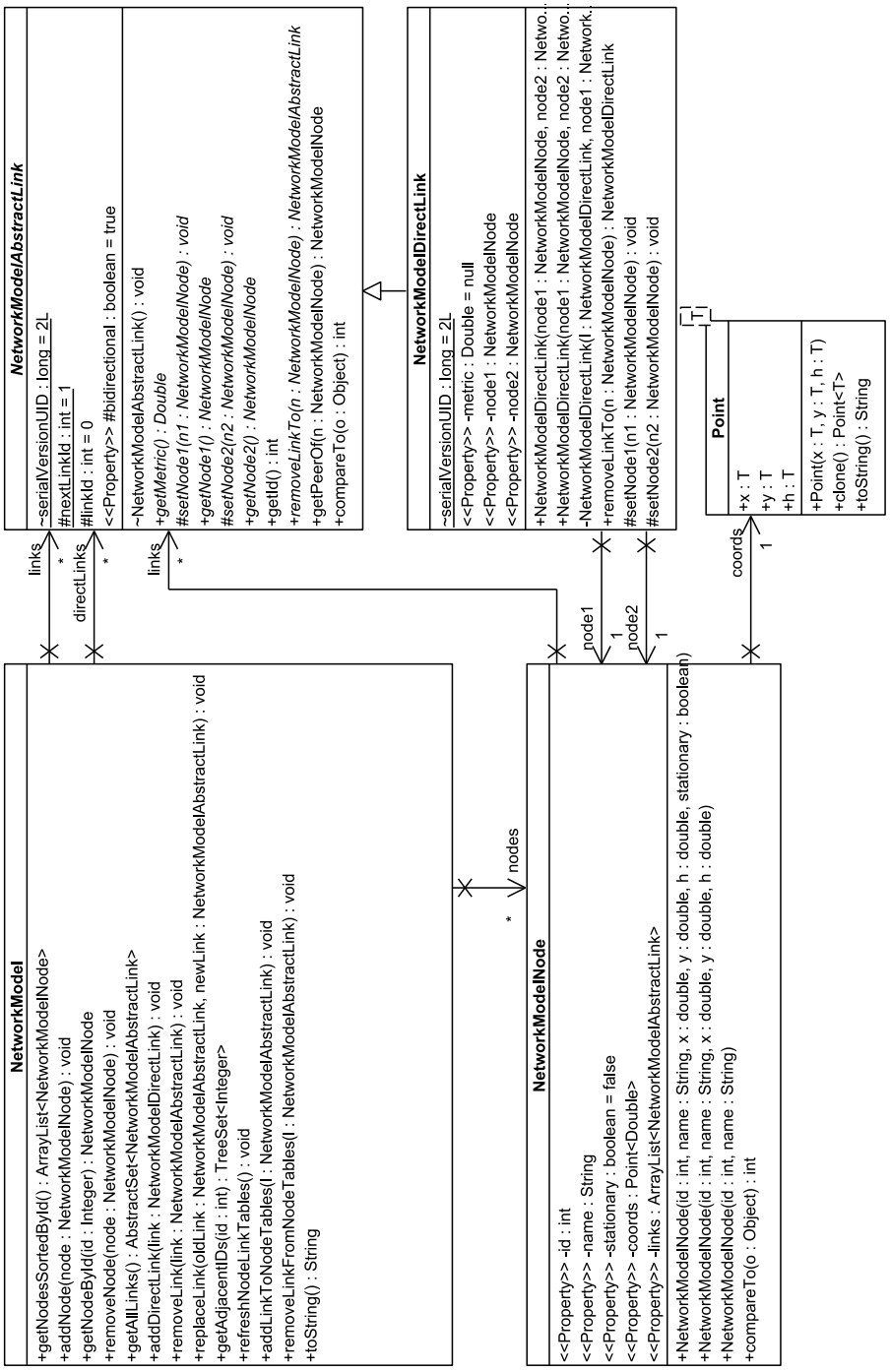




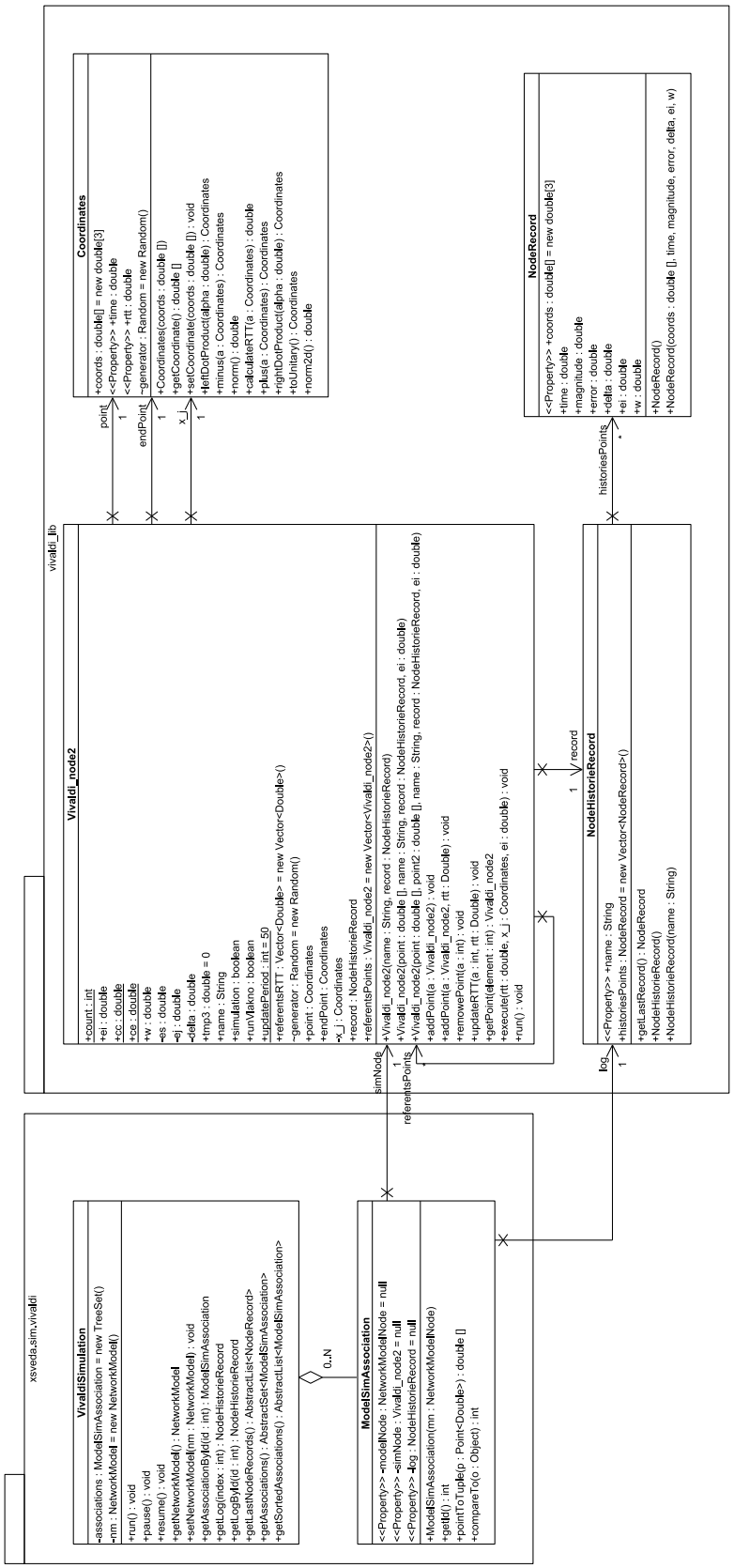
A.3 Diagram hlavních tříd aplikace



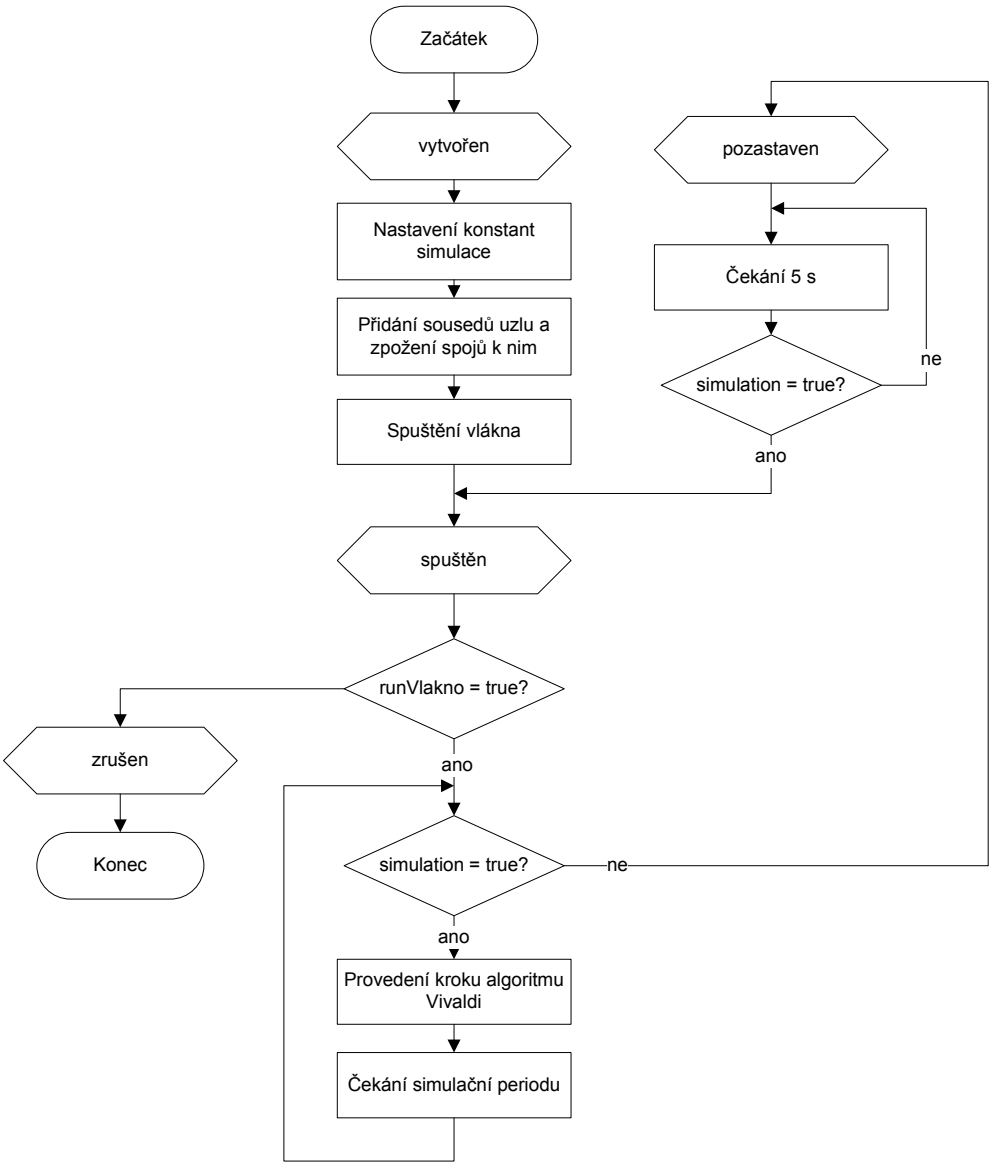
A.4 Diagram tříd modelu sítě



A.5 Vazby s knihovnou vivaldi-lib



A.6 Životní cyklus objektu Vivaldi_node2



B PŘEHLEDOVÝ DIAGRAM KNIHOVNY VIVALDI-LIB

Diagram byl převzat z [26].

C OBSAH PŘILOŽENÉHO CD

Adresář	Obsah adresáře
`aplikace`	aplikace VivaldiMonitor a pomocné programy
`vivaldi_monitor`	aplikace VivaldiMonitor v obou verzích (`vmon.jar` – s adaptivním ČK, `vmon_f.jar` – s konstantním ČK)
`pomocne`	pomocné programy zmíněné v textu
`diagramy`	diagramy použité v práci (plná velikost]
`matlab`	ukázka skriptu pro zpracování exportu dat simulace
`simulace`	soubory simulací sady Meridian
`meridian`	zdrojová data sady Meridian
`modely`	modely použité pro simulace
`vysledky`	snímky grafů a uložené tabulky chyb spojů
`text_diplomove_pr`	text diplomové práce a povinné dokumenty
`zdrojove_kody`	zdrojové kódy programů z adresáře `aplikace`
`VivaldiMonitor`	aplikace VivaldiMonitor pro adaptivní ČK
`VivaldiMonitorF`	aplikace VivaldiMonitor pro konstantní ČK
`knihovny`	použité knihovny
`pomocne`	zdrojové kódy pomocných programů