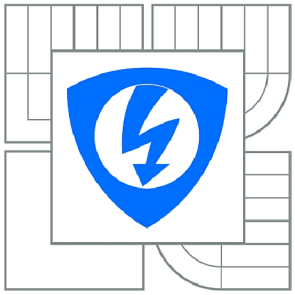
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

SIMULACE LOKALIZACE IP STANIC POMOCÍ ALGORITMŮ VIVALDI A GNP

SIMULATION OF IP NODE LOCALIZATION, USING VIVALDI AND GNP ALGORITHMS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

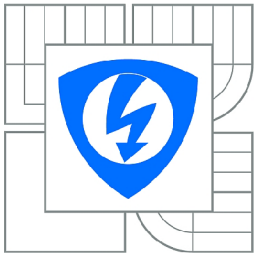
AUTOR PRÁCE — Bc. PETER SULÍK

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAKUB MÜLLER

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Peter Sulík	***ID:***	78325
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Simulace lokalizace IP stanic pomocí algoritmů Vivaldi a GNP

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je vypracovat simulační knihovnu, které bude schopna na základě definovaných vstupních dat simulovat a vyhodnotit proces lokalizace uzlů pomocí algoritmu Vivaldi a GNP. Výsledky by měly být vhodně prezentovány formou dat a grafů.

DOPORUČENÁ LITERATURA:

[1] T. S. Eugene Ng and Hui Zhang, "A Network Positioning System for the Internet", USENIX Annual Technical Conference 2004, Boston, MA, June 2004
[2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In Proceedings of SIGCOMM'04, August 2004.

Termín zadání:	7.2.2011	***Termín odevzdání:***	26.5.2011

Vedoucí práce: Ing. Jakub Müller

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Práca pojednáva o problematike sieťových súradnicových systémov. V prvej kapitole sa venuje ich stručnému rozboru a požiadavkám na ne kladeným. Ďalej sa zaoberá vlastnosťami algoritmu Vivaldi a popisuje jeho tri verzie: centralizovaný algoritmus, algoritmus s konštantným a adaptívnym časovým krokom. Následne popisuje systém GNP a jednotlivé kroky Nelder-Meadovej metódy pre lokalizovanie uzlov. Následne je prezentovaná vyvinutá simulačná knižnica s vlastným grafickým užívateľským rozhraním pre testovanie týchto algoritmov, ktorá je schopná pracovať s databázou odoziev RTT medzi jednotlivými členmi IP siete. V závere sú zhodnotené simulácie vykonané na dátach zo siete PlanetLab.

KĽÚČOVÉ SLOVÁ

Vivaldi, GNP, Pharos, Myth, PCoord, Java, Sieťové súradnicové systémy, PlanetLab

ABSTRACT

The work deals with the issues of network coordinate systems. The first chapter is devoted to a brief analysis of this systems and requirements placed on them. Furthermore, it deals with features of Vivaldi algorithm and its three versions: a centralized algorithm, algorithm with constant and adaptive time step. Subsequently it describes the GNP system and eash step of Nelder-Mead method for nodes localization. Next to it a developed simulation library is presented with its own graphical user interface for testing of these algorithms, which is capable of handling RTT responses database amongst individual IP network members. Simulations performed on data from PlanetLab network are evaluated in the conclusion.

KEYWORDS

Vivaldi, GNP, Pharos, Myth, PCoord, Java, Network coordinate systems, PlanetLab

SULÍK, Peter *Simulace lokalizace IP stanic pomocí algoritmů Vivaldi a GNP:* diplomová práca. Brno: Vysoké učení technické v Brne, Fakulta elektrotechniky a komunikačných technológií, Ústav telekomunikácií, 2011. Vedúci práce bol Ing. Jakub Müller

PREHLÁSENIE

Prehlasujem, že svoju diplomovú prácu na tému „Simulace lokalizace IP stanic pomocí algoritmů Vivaldi a GNP“ som vypracoval samostatne pod vedením vedúceho diplomovej práce a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, hlavne som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušenia ustanovenia §11 a nasledujúceho autorského zákona č. 121/2000 Sb., vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia §152 trestného zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

POĎAKOVANIE

Ďakujem vedúcemu diplomovej práce Ing. Jakubovi Müllerovi za veľmi užitočnú metodickú pomoc a cenné rady pri spracovaní diplomovej práce.

Brno ....

(podpis autora)

OBSAH

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ÚVOD

Dosiahnutie vysokého výkonu je kľúčovým aspektom pri budovaní globálne distribuovateľných sieťových služieb a aplikácií ako webhostingové služby s distribuovaným obsahom, či P2P zdieľanie súborov. Dôvodom je, že tieto systémy majú veľkú flexibilitu pri voľbe komunikačných ciest a musia ich vyberať inteligentne s ohľadom na výkon siete pretože Internet je veľmi rôznorodý priestor. Napríklad v P2P aplikácii na zdieľanie súborov klient v ideálnom prípade chce poznať dostupnú šírku pásma medzi ním samotným a všetkými peermi, ktorí vlastnia žiadaný súbor. Aj keď dynamické výkonové charakteristiky siete ako latencia a dostupná šírka pásma sú pre aplikácie najviac relevantné a môžu byť zmerané na vyžiadanie (on-demand), tak kvôli obrovskému množstvu koncových ciest, ktoré treba pri distribuovaných systémoch posúdiť, sú merania nepraktické, pretože sú nákladné a časovo náročné.

Sľubným prístupom ako preklenúť priepasť medzi optimalizáciou výkonu a škálovateľnosťou siete, je odhadovať sieťovú vzdialenosť (t.j., oneskorenie prenosu a RTT) medzi stanicami a použiť ju ako hlavný faktor metriky na zredukovanie alebo eliminovanie potreby sieťových meraní na vyžiadanie. Z toho dôvodu je kritickým problémom vypracovanie takej techniky, ktorá dokáže predpovedať sieťovú vzdialenosť presne, škálovateľne a v prijateľnom časovom intervale.

Táto diplomová práca si kladie za cieľ oboznámiť o niekoľkých z dostupných algoritmov na odhad latencie: Vivaldi [1], GNP [7], Pharos [4], Myth [5] a PCoord [10]. Ďalším cieľom je vytvorenie simulačnej knižnice, ktorá dokáže pracovať s databázou RTT odoziev nameraných v sieti PlanetLab, a ktorá je schopná vyhodnocovať proces lokalizácie uzlov.

Prvá kapitola popisuje umelé súradnicové systémy a požiadavky na ne. Druhá kapitola bližšie oboznamuje o algoritme Vivaldi a jeho vlastnostiach. Tretia kapitola hovorí o algoritme GNP. Štvrtá kapitola pojednáva o ďalších systémoch na predikciu latencie využívajúce umelé súradnicové systémy: Pharos, Myth a PCoord. V piatej kapitole je popísaná vyvinutá simulačná knižnica pre prácu s RTT hodnotami zo siete PlanetLab a následne sú v šiestej kapitole tieto dáta použité na demonštrovanie funkčnosti vyvinutej knižnice. Nakoniec sú v závere zhrnuté a vyhodnotené výsledky z vykonaných simulácií, a je prediskutovaných niekoľko návrhov na zlepšenie vierohodnosti predikcie latencie.

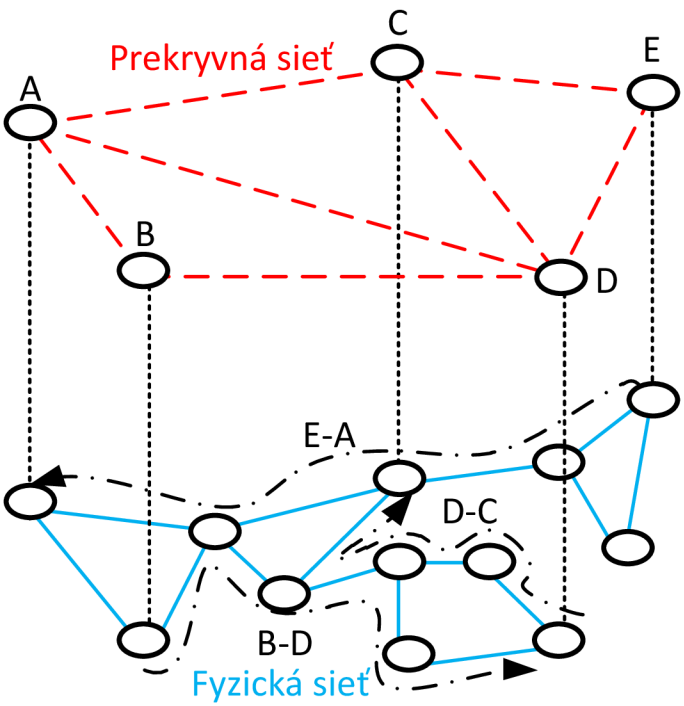
1 UMELÉ SÚRADNICOVÉ SYSTÉMY

Umelé súradnicové systémy umožňujú uzlom v Internete odhadovať latencie k ostatným uzlom. Uzly si vypočítajú pozície v takom súradnicovom priestore aby vzdialenosť medzi danými umelými súradnicami odhadovala oneskorenie medzi nimi v Internete. Tým pádom, ak si uzol *x* zistí súradnice uzlu *y*, nemusí vykonávať explicitné merania v sieti na určenie hodnoty oneskorenia k uzlu *y*. Naopak, vzdialenosť medzi uzlami *x* a *y* v súradnicovom priestore je presná predikcia hodnoty RTT.

Výsledná funkčnosť umelých súradníc závisí od vlastností Internetu. Ak je napríklad na linkách v Internete len minimálne oneskorenie a ak existuje dostatočné množstvo prepojení, že medzi každými dvoma uzlami je zhruba jedno fyzické prepojenie, tak je s najväčšou pravdepodobnosťou možné, že umelé súradnice, ktoré napodobňujú zemepisnú dĺžku a šírku, budú správne odhadovať latenciu.

Avšak tieto vlastnosti sú len približné. Len niekoľko dvojíc uzlov na dlhej trase je priamo spojených, poskytovatelia Internetového pripojenia sú rozmiestnení len na limitovanom počte oblastí, prenosový čas a operácie vykonávané smerovačmi pri smerovaní paketu vytvárajú oneskorenie. Z týchto dôvodov sa pakety často odchyľujú od dlhých trás. Tým pádom je nemožné vybrať 2-rozmerný súradnicový priestor pre popisovanie pozícií uzlov, lebo môže dochádzať k veľkému skresleniu latencií. Umelý súradnicový systém tak musí mať spôsob ako vybrať také súradnice, čo budú minimalizovať predikčné chyby čo najlepšie. Súradnice navyše nemusia byť limitované na dve dimenzie.

Schopnosť predpovedať hodnotu RTT bez predošlej komunikácie dovoľuje systémom používať približné informácie pre zvýšenie výkonu, s menším počtom meraní a réžiou. Súradnicový systém môže byť napríklad použitý na výber z množstva replikovaných dátových serverov z. Je nápomocný najmä keď počet potencionálnych serverov je príliš vysoký alebo množstvo dát na prevzatie je malé. V oboch prípadoch by nebolo praktické zisťovať najprv ktorý server je najbližšie, pretože náklady na toto vynaložené by prevážili nad výhodou inteligentného výberu najbližšieho serveru. Systémy s distribuovaným obsahom alebo zdieľanými súbormi ako BitTorrent [9] či KaZaA sú príkladmi systémov ponúkajúcich veľké množstvo duplicitných serverov. DNS (Domain Name System) je na druhej strane príkladom takého systému, ktorý ponúka priemerné množstvo duplicitných serverov, ale každý kus prevzatých dát je malý. Všetky vyššie spomenuté aplikácie by mohli ťažiť zo sieťových súradníc.



Obr. 1.1: Príklad rozdielu logickej topológie siete od fyzickej

1.1 Požiadavky na umelé súradnicové systémy

Návrh umelého súradnicového systému pre použitie v distribuovaných aplikáciách s veľkým množstvom staníc zahŕňa nasledovné problémy:

- Nájdenie takého metrického priestoru, ktorý simuluje Internet s malou chybou. Vhodný priestor by si mal poradiť s prekážkami ako je smerovanie v Internete, prenosová doba alebo tvorba front.
- Rozšírenie na veľký počet staníc. Umelé súradnicové systémy majú výhodu len v aplikáciách s veľkým počtom staníc. Pri malom počte je priame meranie RTT výhodnejšie.
- Decentralizovaná implementácia. Množstvo novovznikajúcich aplikácií, ako napríklad P2P aplikácie, sú distribuované a symetrické a neobsahujú špeciálne uzly, ktoré by mohli spoľahlivo plniť funkciu orientačných bodov.

- Minimalizovanie signalizačného prenosu. V ideálnom prípade by umelý súradnicový systém nemal tvoriť dodatočný prenos, naopak, by mal byť schopný všetky potrebné informácie získať z dostupných informácií jednotlivej aplikácie.
- Adaptácia na zmeny v sieti. Relatívna pozícia staníc v sieti sa môže meniť z dôvodu rekonfigurácie siete. Systém by mal byť schopný periodicky upravovať súradnice staníc aby mohol reagovať na podobné zmeny.

1.2 Štruktúra umelých súradníc

Algoritmy si môžu vyberať štruktúru súradníc a funkciu vzdialenosti, ktorá určuje odhadovanú latenciu z dvoch súradníc. Súradnice by mali byť kompaktné a malo by byť jednoduché vypočítať odhadovanú hodnotu RTT na základe daných dvoch súradníc. Najjednoduchším spôsobom je použitie n-rozmerných súradníc s štandardnou funkciou Euklidovskej vzdialenosti. Sférická, toroidná, hyperbolická a iné štruktúry súradníc boli tiež navrhované. Tieto súradnicové systémy používajú alternatívne funkcie vzdialenosti v nádeji, že výsledná hodnota odhadovanej latencie bude presnejšia. V tejto práci s ohľadom na dostupnú databázu zmeraných hodnôt RTT som pracoval s 2-rozmerným súradnicovým systémom.

1.3 Algoritmus vivaldi

Vivaldi [1][2] je jednoduchý decentralizovaný algoritmus, nenáročný na systémové prostriedky, ktorý priraďuje umelé súradnice staniciam tak, aby vzdialenosť medzi dvojicou staníc presne odhadovala komunikačnú latenciu medzi danou dvojicou uzlov.

Je to tiež plne distribuovateľný algoritmus, ktorý nepotrebuje žiadnu fixnú infraštruktúru či špeciálne stanice. Je taktiež efektívny: novo pripojený uzol môže vypočítať presné súradnice na základe RTT informácií, ktoré získal od dostatočného, no malého, množstva susedných uzlov čím sa predchádza zbytočnému kontaktovaniu ďalšieho množstva susedných uzlov [6]. Keďže jeho požiadavky na komunikáciu sú minimálne, môže potrebné informácie získavať technikou piggy-backingu[1] na aplikačnej vrstve a takto byť rozšíriteľný na množstvo staníc.

[1] piggy-backing – je obojsmerná technika prenosu dát v sieťovej vrstve. Do dátového rámca pridáva potvrdenie (ACK - acknowledge) o prijatí dát, namiesto toho aby sa používal individuálny rámec.

V algoritme Vivaldi si každý uzol počíta vlastné súradnice. Vždy keď nejaký uzol komunikuje s iným uzlom, tak zmeria hodnotu latencie a upraví svoje súradnice tak, aby chyba medzi zmeranou a odhadovanou hodnotou latencie (predikčná chyba) bola minimálna. Od užívateľa je vyžadované aby nastavil len jediný parameter, ktorý popisuje o akú maximálnu hodnotu môže uzol upraviť svoje súradnice pri získaní nových vzoriek latencií. Tento parameter z veľkej časti nezávisí na vstupe algoritmu a môže byť nastavený tak, aby zaisťoval presnosť súradníc na úkor doby konvergencie siete.

1.3.1 Predikčná chyba

Nech L_{ij} je aktuálna hodnota RTT medzi dvoma uzlami i a j, a nech x_i sú súradnice priradené uzlu i. Môžeme takto definovať chybu v súradniciach použitím funkcie celkovej kvadratickej chyby:

$$E = \sum_i \sum_j \left(L_{ij} - \|x_i - x_j\|\right)^2 \tag{1.1}$$

Kde $\|x_i - x_j\|$ je vzdialenosť medzi súradnicami uzlu i a j vo zvolenom súradnicovom priestore. Minimalizácia funkcie kvadratickej chyby u algoritmu Vivaldi je vybraná z toho dôvodu, lebo má analógiu k posunutiu v masovom strunovom systéme. Minimalizovanie chyby v sieti strún je ekvivalentné minimalizovaniu funkcie kvadratickej chyby.

1.3.2 Centralizovaný algoritmus

Najprv je vhodné popísať jednoduchý, centralizovaný algoritmus, ktorý dokáže minimalizovať rovnicu (2.1). Vivaldi je distribuovanou verziou tohto algoritmu. Vzhľadom k výberu celkovej kvadratickej chyby, simulovanie pružinovej siete vytvára súradnice, ktoré minimalizujú túto chybu.

Keďže celková kvadratická chyba je ekvivalentná potencionálnej energii pružnosti, môžeme ju minimalizovať simulovaním pohybov uzlov pod vplyvom síl pružnosti. Konfigurácia s minimálnou energiou pružnosti korešponduje s minimálnou chybou pridelených súradníc a nie je garantované, že simulácia nájde globálne minimum. Môže sa stať, že systém zotrvá v lokálnom minime.

Môžeme definovať F_{ij} ako vektor sily, ktorým pružina medzi uzlami *i* a *j* pôsobí na uzol *i*. Z Hookovho zákona platí, že sila F je:

$$F_{ij} = \left(L_{ij} - \left\|x_i - x_j\right\|\right) \times u\left(x_i - x_j\right) \tag{1.2}$$

Skalárna veličina $\left(L_{ij} - \left\|x_i - x_j\right\|\right)$ je posunutie pružiny od zvyšných uzlov. Táto veličina udáva veľkosť sily, ktorou pôsobí pružina na uzly *i* a *j*. Jednotkový vektor $u\left(x_i - x_j\right)$ udáva smer sily od uzlu *i*. Násobenie jednotkového vektoru veľkosťou sily vypočítanou vyššie udáva vektor sily, ktorým pôsobí pružina na uzol *i*. Výsledná sila pôsobiaca na uzol *i* je sumou síl z ostatných uzlov:

$$F_i = \sum_{j \neq i} F_{ij} \tag{1.3}$$

Pre simulovanie evolúcie pružinovej siete algoritmus uvažuje s malými časovými intervalmi. V každom intervale algoritmus posunie uzlom x_i o malú vzdialenosť v súradnicovom priestore v smere vektoru sily F_i a potom prepočíta všetky sily. Súradnice na konci časového intervalu sú:

$$x_i = x_i + F_i \times t \tag{1.4}$$

Kde *t* je dĺžka časového intervalu. Veľkosť *t* udáva ako ďaleko sa uzol posunie v jednotlivom časovom intervale. Pri navrhovaní algoritmu Vivaldi je nájdenie vhodného *t* veľmi dôležité.

Algoritmus 1: Centralizovaný algoritmus

```
// Vstup: Matica latencií L a inicializačné súradnice uzlu x
// Výstup: Spresnené súradnice uzlu x
while (chyba (L, x) > tolerancia)
    foreach i
        F = 0
        foreach j
            // Vypočítaj chybu/silu tejto pružiny
            L_ij − ||x_i − x_j||
```

// Pridaj vektor sily tejto pružiny do výslednej sily

$$F = F + e \times u(x_i - x_j)$$

// Urob malý krok v smere sily

$$x_i = x_i + t \times F$$

Vyššie je znázornený pseudokód centralizovaného algoritmu. Pre každý uzol *i* v systéme vypočíta algoritmus silu pružnosti pôsobiacu na každý uzol pripojený k uzlu *i* a spočíta túto silu s celkovou silou pôsobiacou na uzol *i*. Po tom ako je vypočítaná výsledná sila sa uzol *i* posunie o malú vzdialenosť v jej smere. Tento proces sa opakuje pokým systém skonverguje na také súradnice, ktoré majú vhodnú predikčnú chybu.

1.3.3 Distribuovaný algoritmus s konštantným časovým krokom

Centralizovaný algoritmus popísaný vyššie počíta súradnice pre všetky uzly z všetkých nameraných hodnôt RTT. V distribuovanej verzii tohto algoritmu každý uzol priebežne počíta a upravuje svoje súradnice len na základe zmeraných RTT hodnôt k hŕstke susedných uzlov. Každý uzol, ktorý je súčasťou Vivaldi algoritmu simuluje vlastný pohyb v pružinovom systéme a spravuje vlastné súradnice. Kedykoľvek sa uzol snaží komunikovať s iným uzlom, zmeria hodnotu RTT k danému uzlu a informuje sa o jeho súradniciach.

Vstupom distribuovaného Vivaldi algoritmu je práve takáto sekvencia RTT prvkov. Odozvou k týmto prvkom, je posunutie uzlu o malý časový krok korešpondujúcou pružinou. Každé ďalšie posunutie redukuje chybovosť uzlu s ohľadom na ostatné uzly v systéme. Ako uzly priebežne spolu komunikujú, konvergujú k takým súradniciam, ktoré pomerne presne predpovedajú RTT vzdialenosti.

Keď sa uzol *i* so súradnicami x_i dozvie o uzle *j* so súradnicami x_j a príslušnej RTT hodnote k nemu, aktualizuje svoje súradnice podľa nasledovného pravidla:

$$x_i = x_i + \delta \times \left(L_{ij} - \left\|x_i - x_j\right\|\right) \times u(x_i - x_j) \tag{1.5}$$

Toto pravidlo je identické k jednotlivým silám vypočítaným vo vnútornom cykle centralizovaného algoritmu.

Algoritmus 2: Distribuovaný algoritmus s konštantným časovým krokom

// Vstup: Nameraná rtt vzdialenosť a súradnice x_j vzdialeného uzlu j

// Výstup: Aktualizované súradnice uzlu i

// Vypočítanie chyby pre tento prvok

$e = L_{ij} - \|x_i - x_j\|$

// Nájdenie smeru sily spôsobenej chybou

$dir = u(x_i - x_j)$

// Vektor sily je priamo úmerný chybe

$f = dir \times e$

// Posun o malý krok v smere sily

$x_i = x_i + \delta \times dir$

Pseudokód pre tento distribuovaný algoritmus je popísaný vyššie. Je volaný kedykoľvek dôjde k novému meraniu RTT. Je použitá nová hodnota RTT ako aj súradnice vzdialeného uzlu. Procedúra najprv vypočíta chybu v svojej momentálnej predikcii k danému vzdialenému uzlu. Lokálny uzol sa potom posunie smerom k alebo od vzdialeného uzlu na základe veľkosti tejto chyby. V ďalšom kroku sa vypočíta smer, v ktorom by sa uzol mal posunúť. A nakoniec sa uzol posunie o kus v smere vzdialeného uzlu použitím konštantného časového kroku.

1.3.4 Distribuovaný algoritmus s adaptívnym časovým krokom

Hlavným problémom v implementácii algoritmu Vivaldi je zaistenie konvergencie uzlov na také súradnice, ktoré čo najpresnejšie odhadujú RTT vzdialenosť. Rýchlosť konvergencie riadi časový krok *δ*. Príliš veľké hodnoty *δ* zapríčiňujú, že Vivaldi upravuje súradnice v dlhých krokoch. Avšak ak všetky uzly používajú príliš veľké hodnoty *δ*, tak výsledkom je obyčajne oscilácia a neschopnosť konvergencie na použiteľné súradnice.

Pre dosiahnutie rýchlej konvergencie a vyhnutiu sa oscilácii Vivaldi obmieňa *δ* v závislosti akú má uzol istotu o správnosti jeho súradníc. V priebehu ako uzol zisťuje svoju približnú polohu v sieti (napríklad pri prvotnom prihlásení) používa väčšie hodnoty *δ*, ktoré mu umožňujú rýchlejšie meniť a dosiahnuť približnú orientačnú polohu. Následne, menšie hodnoty *δ* mu pomôžu ešte viac spresniť svoju polohu.

Ďalším problémom sú uzly, ktoré majú vysokú chybovosť vlastných súradníc. Ak uzol *i* komunikuje s nejakým uzlom, ktorý má také súradnice čo odhadujú RTT vzdialenosť s veľkou chybou, tak akákoľvek aktualizácia ktorú uzol *i* vykoná na základe daných chybových súradníc s veľkou pravdepodobnosťou zvýši predikčnú chybu. Z toho dôvodu je časový krok implementovaný nasledovne:

(1.6)

$$\delta = c_c \times \frac{chyba\ lokálneho\ uzlu}{chyba\ lokálneho\ uzlu + chyba\ vzdialeného\ uzlu}$$

kde c_c je konštantná časť chyby ($c_c < 1$). Takto definovaný časový krok je odolný voči uzlom s vysokou chybovosťou, schopný rýchlej konvergencie a nenáchylný k oscilácii. Je vyžadované aby uzly udržovali stály odhad nad presnosťou svojich súradníc. Dosahujú to opakovaným porovnávaním vzoriek RTT s odhadovanou hodnotou RTT a udržovaním priemeru relatívnych chýb (pomer absolútnej chyby k latencii). Váha každej vzorky je určená pomerom odhadovanej relatívnej chyby lokálneho uzlu k vzorkovanému uzlu.

Pseudokód pre distribuovaný algoritmus Vivaldi s adaptívnym časovým krokom je zobrazený nižšie. Táto procedúra počíta váhu vzorky na základe lokálnej a vzdialenej chyby. Algoritmus musí tiež zaznamenávať lokálnu relatívnu chybu a to použitím váženého priemeru. Zvyšok algoritmu je zhodný s verziou s konštantným časovým krokom.

Algoritmus 3: Distribuovaný algoritmus s adaptívnym časovým krokom

// Vstup: nameraná vzdialenosť rtt, súradnice x_j a chyba e_j vzdialeného uzlu j

// Výstup: aktualizované súradnice x_i a chyba e_i uzlu i

// Konštanty c_c a c_e sú ladiace parametre

// Vyváženie lokálnej a vzdialenej chyby

$$w = e_i/(e_i + e_j)$$

// Výpočet relatívnej chyby tejto vzorky

$$e_s = \left|\|x_i - x_j\| - rtt\right|/rtt$$

// Aktualizácia váženého priemeru lokálnej chyby

$$e_i = e_s \times c_e \times w + e_i \times (1 - c_e \times w)$$

// Aktualizácia lokálnych súradníc

$$\delta = c_c \times w$$

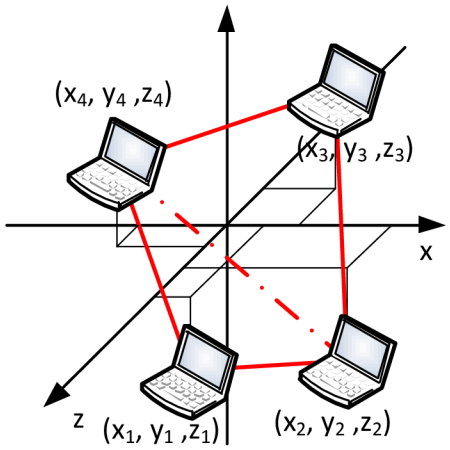
$$x_i = x_i + \delta \times \left(rtt - \|x_i - x_j\|\right) \times u(x_i - x_j)$$

1.4 GNP (Global network positioning)

Hlavnou myšlienkou algoritmu GNP [7] je modelovať Internet ako geometrický priestor (napr. 3-rozmerný Euklidovský priestor) a popísať pozíciu ktorejkoľvek stanice bodom

v tomto priestore. Sieťová vzdialenosť medzi dvojicou staníc je potom odhadovaná modelom geometrickej vzdialenosti medzi nimi.

V prvom kroku malý distribuovaný súbor staníc, v algoritme GNP nazývaný orientačné body (Landmarks), vypočíta svoje vlastné súradnice vo zvolenom súradnicovom priestore. Súradnice orientačných bodov slúžia ako referenčný rámec a sú šírené na ktorékoľvek stanice, ktoré sa chcú zapojiť. V druhom kroku si môže ktorákoľvek stanica vypočítať vlastné súradnice, ktoré sú relatívne k súradniciam orientačných bodov.



Obr. 1.2: Geometrický model Internetu

1.4.1 Orientačné body (Landmarks)

Je potrebné definovať model Internetu ako geometrický priestor *S*. Súradnice staníc *H* v tomto priestore je možné definovať ako c^S_H, funkciu vzdialenosti, ktorá operuje na týchto súradniciach ako $f^S()$ a vypočítanú vzdialenosť medzi uzlami H_1 a H_2 ako $d^S_{H_1 H_2}$.

V geometrickom priestore je potrebné určiť malý distribuovaný súbor staníc ako orientačné body, ktoré budú poskytovať súbor orientačných súradníc potrebných na orientovanie sa zvyšných staníc v priestore *S*. Orientačné body jednoducho zmerajú RTT hodnoty medzi sebou a z miním vytvoria spodnú polovicu matice vzdialeností *N x N* (matica je symetrická podľa diagonály). Zmerané vzdialenosti medzi stanicami H_1 a H_2 možno označiť ako $d_{H_1 H_2}$. Za použitia zmeraných vzdialeností, jedna zo staníc, pravdepodobne orientačný uzol, vypočíta súradnice ostatných orientačných bodov v priestore *S*. Cieľom je nájsť taký súbor súradníc pre *N* orientačných bodov, aby celková

chyba medzi zmeranou vzdialenosťou a vypočítanou vzdialenosťou v priestore *S* bola čo najmenšia. Z uvedeného vyplýva, že je potrebné nájsť minimalizáciu nasledovnej funkcie:

(1.7)

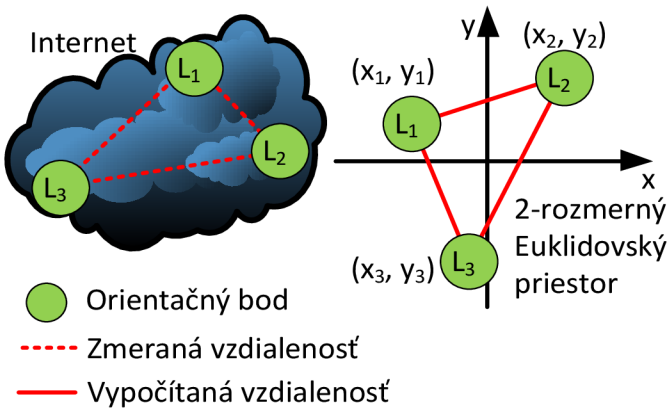
$$f_{obj1}\left(c_{L_i}^S, \ldots, c_{L_N}^S\right) = \sum_{L_iL_j \in \{L_1,\ldots,L_N\} | i<j} \varepsilon\left(d_{L_iL_j}, d_{L_iL_j}^S\right)$$

Kde ε je funkcia na meranie chyby, ktorá môže byť napríklad jednoduchá kvadratická chyba:

(1.8)

$$\varepsilon\left(d_{H_1H_2}, d_{H_1H_2}^S\right) = \left(d_{H_1H_2} - d_{H_1H_2}^S\right)^2$$

alebo iná sofistikovaná funkcia na meranie chyby. Spôsob akým je vo funkcii meraná chyba má kritický dopad na presnosť odhadovanej vzdialenosti. Algoritmov na nájdenie extrémov tejto funkcie existuje viac. V pôvodnej štúdii bola použitý Nelder-Meadova metóda. Ako náhle sú vypočítané súradnice orientačných bodov, dochádza k ich rozšíreniu medzi bežné stanice spolu s identifikátorom použitého geometrického priestoru a korešpondujúcou funkciou vzdialenosti.



Obr. 1.3: Operácie vykonávané orientačnými bodmi

1.4.2 Stanice (Hosts)

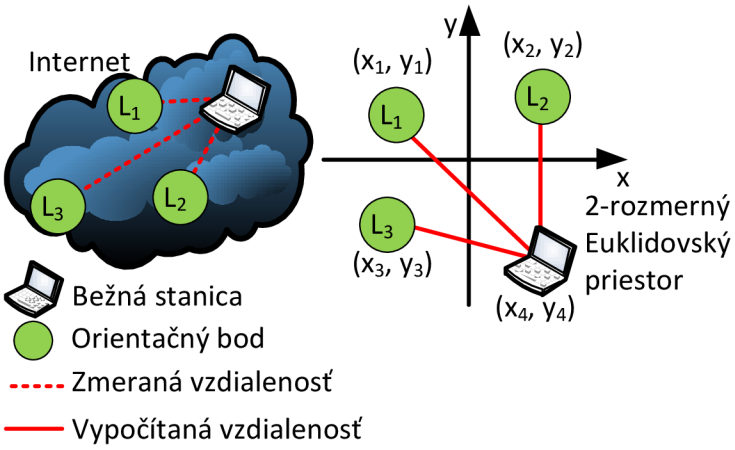
Za použitia súradníc orientačných bodov v geometrickom priestore si každá stanica odvodí vlastné súradnice. To dosiahne tak, že zmeria RTT hodnotu k *N*

orientačným bodom použitím ICMP ping správ, z ktorých si zoberie minimálne hodnoty a stanoví ich za vzdialenosť k danému orientačnému bodu. V tejto fáze sú orientačné body kompletne pasívne a len odpovedajú na prichádzajúce ICMP ping správy. Stanice na základe *N* meraní latencií k orientačným uzlom dokážu vypočítať svoje súradnice, ktoré minimalizujú celkovú chybu medzi zmeranou a vypočítanou vzdialenosťou k orientačným bodom. Formálne, je potrebné minimalizovať nasledovnú funkciu:

(1.9)

$$f_{obj2}(c_H^S) = \sum_{L_i \in \{L_1,..,L_N\}} \varepsilon\left(d_{HL_i}, d_{HL_i}^S\right)$$

Kde ε je znovu funkciou chyby merania. Nájdenie extrémov tejto funkcie je možné dosiahnuť rovnakými algoritmami ako pri orientačných bodoch.



Obr. 1.4: Operácie vykonávané bežnými stanicami

1.4.3 Nelder-Meadova metóda

Nelder-Meadova metóda, tiež nazývaná Simplex-Downhill, je simplexová metóda určená na nájdenie lokálneho minima funkcie o niekoľkých neznámych. Jej vynájdenie je pripisované pánom J. A. Nelder a R. Mead. Pre dve neznáme je simplexom trojuholník, a metóda porovnáva funkčné hodnoty troch vrcholov trojuholníka. Najhorší vrchol (s najväčšou funkčnou hodnotou v bode so súradnicami *x* a *y*) je nahradený novým vrcholom a v ďalšom kroku sa sformuje nový trojuholník. Tento proces generuje sekvenciu trojuholníkov (s rozdielnym tvarom), pre ktoré sú funkčné hodnoty vo vrcholoch stále

menšie a menšie. Takto sa zmenšuje veľkosť trojuholníkov až na minimum, keď môžeme predpokladať, že sme našli súradnice globálneho minima. Algoritmus definuje pojem simplex, čo je univerzálny trojuholník v N-rozmernom priestore a dokáže nájsť minimum funkcie o N neznámych. Je efektívny a výpočtovo kompaktný.

Počiatočný trojuholník BGW

Nech f_{xy} je funkcia, ktorá má byť minimalizovaná. Na začiatku sú dané tri vrcholy trojuholníka: $V_k(x_k,y_k)$ pre k = 1, 2, 3. Hodnota funkcie je následne hľadaná v každom z troch vrcholov $z_k = f(x,y)$ pre k = 1, 2, 3. Následne sú indexy poprehadzované tak, aby platilo $z_1 < z_2 < z_3$. Zaužívané je označenie BGW tak, aby B (best) bol najlepší vrchol s najmenšou funkčnou hodnotou, W (worst) najhorší vrchol s najvyššou funkčnou hodnotou.

Stredový bod M

Je potrebné definovať z dôvodu použitia vo všetkých zmenách vykonávaných so simplexným trojuholníkom. Je nájdený ako stredový bod ležiaci na priamke medzi bodmi B a G a to spriemerovaním súradníc:

(1.10)

$$\vec{M} = \frac{1}{2}\left(\vec{B} + \vec{G}\right) = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$$

Po nájdení počiatočného trojuholníka a bodu W je potrebné upraviť jeho pozíciu tak, aby hodnota v novom bode bola nižšia než v pôvodnom bode W. K tomu slúžia štyri úpravy trojuholníku, resp. bodu W a to: zrkadlenie, roztiahnutie, zmrštenie a stiahnutie k bodu B.

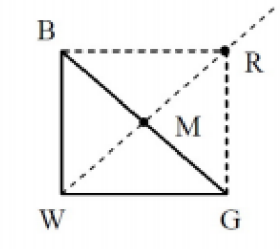
Zrkadlenie pomocou bodu R

Hodnota funkcie klesá po priamke v smere od bodu W k bodu B a taktiež k bodu G. Tým pádom je možné predpokladať, že hodnota funkcie nadobúda menšie hodnoty v bodoch vzdialených od vrcholu W na opačnej strane priamky medzi B a G. Je teda vhodné nájsť testovací bod v tomto priestore. Dá sa tak učiniť zrkadlením vrcholu W cez priamku BG. Najprv je však potrebné nájsť bod M ako stred priamky BG a následne preložiť polpriamku cez bod M z vrcholu W. Bod R leží na tejto polpriamke v rovnakej

vzdialenosti od bodu M ako W. Celý proces je znázornený na obrázku a spôsob akým sa vypočíta pozícia bodu R v rovnici:

(1.11)

$$\vec{R} = \vec{M} + \left(\vec{M} - \vec{W}\right) = 2\vec{M} - \vec{W}$$



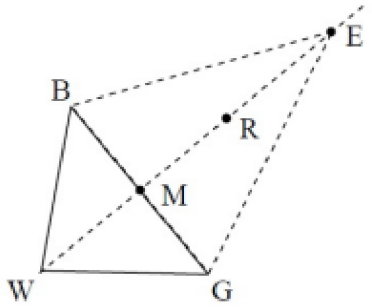
Obr. 1.5: Úprava zrkadlením

Roztiahnutie pomocou bodu E

Ak je funkčná hodnota v R menšia ako funkčná hodnota vo W, bol pohyb v tomto smere správny a smerom k minimu. Dá sa predpokladať, že minimum leží o kúsok ďalej na polpriamke WM. Tak sa položí nová priamka z bodu R k bodu E. Ak je funkčná hodnota v bode E menšia ako funkčná hodnota v bode R, tak bol nájdený lepší vrchol než R. Rovnica pre výpočet E je:

(1.12)

$$\vec{E} = \vec{R} + \left(\vec{R} - \vec{M}\right) = 2\vec{R} - \vec{M}$$



Obr. 1.6: Úprava roztiahnutím

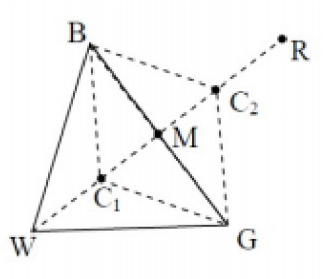
Skrátenie pomocou bodu C

Ak sú funkčné hodnoty v bodoch R a W zhodné, je potrebné otestovať ďalší bod. Je možné predpokladať, že hodnota v bode M je menšia než v bode W, ale ich zámena nie je možná z dôvodu nedodržania trojuholníka. Predpokladajme dva body, C_1 a C_2, na

polpriamke WM. Bod s menšou funkčnou hodnotou bude nazvaný C a bude vytvorený nový trojuholník BGC. Výber medzi bodmi C_1 a C_2 sa zdá byť nezmyselný pre 2-rozmerný prípad, ale je dôležitý vo vyšších dimenziách.

(1.13)

$$\overrightarrow{C_1} = \frac{\left(\vec{W} + \vec{M}\right)}{2} \; alebo \; \overrightarrow{C_2} = \frac{\left(\vec{M} + \vec{R}\right)}{2}$$



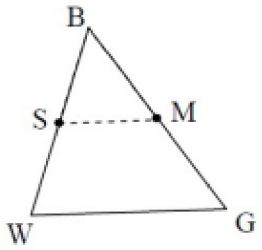
Obr. 1.7: Úprava skrátením

Zmrštenie k bodu B

Ak je funkčná hodnota v bode C vyššia ako v bode W, je potrebné aby boli body G a W posunuté bližšie k bodu B. Bod G je nahradený bodom M a bod W je nahradený bodom S, ktorý je stredom priamky BW.

(1.14)

$$\vec{S} = \frac{\left(\vec{B} + \vec{W}\right)}{2}$$



Obr. 1.8: Úprava zmrštením

Logické rozhodnutia pre každý krok

Výpočtovo efektívny algoritmus vykonáva vyhodnotenia funkcie len keď je to potrebné. V každom kroku je nájdený nový vrchol, ktorý nahradzuje W. Ako náhle je nájdený, sú ďalšie hľadania nepotrebné a iteračný krok je skončený. Pseudokód Nelder-Meadovej metódy je zobrazený nižšie.

Algoritmus 4: Nelder-Mead

If *f(R) < f(G)* **then**
    Vykonaj kód 1 {zrkadlenie alebo roztiahnutie}
else
    Vykonaj kód 2 {zmrštenie alebo stiahnutie k bodu *B*}
end if

BEGIN kód 1
If *f(B) < f(R)* **then**
    zameň *W* za *R* {zrkadlenie}
else
    vypočítaj súradnice *E* a *f(E)*
    if *f(E) < f(B)* **then**
        zameň *W* za *E* {roztiahnutie}
    else
        zameň *W* za *R* {zrkadlenie}
    end if
end if
END kód 1

BEGIN kód 2
If *f(R) < f(W)* **then**
    zameň *W* za *R*
end if
vypočítaj súradnice *C* a *f(C)*
if *f(C) < f(W)* **then**
    zameň *W* za *C* {zmrštenie}
else
    vypočítaj súradnice *S* a *f(S)*
    zameň *W* za *S* {stiahnutie k *B*}
    zameň *G* za *M* {stiahnutie k *B*}
end if
END kód 2

2 INÉ METÓDY PREDIKCIE LATENCIE

2.1 Systém Pharos

V [3][4] bolo zistené, že pre krátke vzdialenosti stúpa chyba predikcie. Tento problém rieši systém zvaný Pharos, ktorý využíva rôzne súradnicové súbory pre rovnaký uzol. Každému uzlu sú priradené viaceré súradnicové súbory, z ktorých niektoré popisujú pozíciu z globálneho meradla a niektoré z pohľadu kratších vzdialeností. Pre jednoduchosť sú v systéme Pharos použité dva súbory súradníc, zatiaľ čo počet súborov sa môže meniť v závislosti na škálovaní dosahu vzdialenosti. V tomto systéme všetky uzly tvoria dve úrovne prekrytia: základnú úroveň prekrytia pre odhad na dlhých linkách a lokálnu úroveň prekrytia pre odhad na krátkych linkách. Z toho vyplýva, že je možné definovať dva typy väzieb: väzby na základnej prekryvnej vrstve ktoré vznikajú medzi uzlami a ich náhodnými susedmi vo vrstve Pharos. A väzby v lokálnej množine, ktoré vznikajú medzi uzlami a ich náhodne vybranými susedmi v lokálnej množine.

V systéme Pharos sa musia uzly pripojiť do základovej vrstvy a zároveň do lokálnej množiny aby mohli získať súradnice v rôznych odhadových úrovniach. Na pripojenie do základovej vrstvy uzly môžu nasledovať procedúru v [1][2] a vytvoriť náhodné väzby k susedom v základovej vrstve. Na vytvorenie lokálnej množiny sa používa metóda podobná binningu[2] a vyberajú sa niektoré uzly, nazývané kotvy, na pomoc pri zhlukovaní uzlov. Akékoľvek uzly, ktoré sú schopné odpovedať na ICMP ping správy môžu slúžiť ako kotvy, podobne ako existujúce DNS servery. Každej kotve je priradený identifikátor označovaný AID. Uzly sprevádzané kotvami sa nasledovným spôsobom zoskupujú do zhlukov. Nový uzol zmeria svoju vzdialenosť ku všetkým kotvám, nájde najbližšiu a pripojí sa do zhluku ktorý vedie. Procedúra pripojenia sa ku zodpovedajúcemu zhluku sa tiež vykonáva podľa a uzol vytvorí náhodné väzby so susedmi v rovnakom zhluku.

Algoritmus Vivaldi je aplikovaný v základovej vrstve ako aj v lokálnej množine. Výsledkom je, že každý uzol v systéme Pharos má dva súbory súradníc. Súradnice vypočítané v základovej vrstve, ktoré sa označujú globálne sieťové súradnice, sa používajú v globálnom meradle. A súradnice vypočítané v zodpovedajúcom zhluku, ktoré sa označujú lokálne sieťové súradnice, pokrývajú menší dosah vzdialeností.

[2] data binning – technika predspracovania dát na zredukovanie minoritných chýb pozorovania. Dáta ktoré spadajú pod spoločný interval (bin), sú reprezentované jedinou, zvyčajne stredovou, hodnotou z intervalu.

Algoritmus uvedený nižšie zobrazuje procedúru pripojenia uzlu A do základovej vrstvy. Uzol A najprv kontaktuje Rendezvous Point (RP) systému Pharos ako v akejkoľvek inej P2P schéme. Po obdŕžaní zoznamu kotiev od RP, zmeria uzol A vzdialenosť ku všetkým ostatným kotvám a nasledovne si vyberie najbližší zhluk do ktorého sa pripojí. Každý uzol periodicky vytvára členské správy (membership messages) pre RP aby oznámil svoju existenciu v systéme. Uzol A sa následne pripojí do základovej vrstvy ako aj lokálnej množiny pomocou protokolu Gossip [11]. Ďalej sa už uzol A môže zúčastňovať výpočtovej procedúry sieťových súradníc v základovej vrstve a lokálnej množine, a môže periodicky aktualizovať svoje súradnice.

Po obdŕžaní globálnych a lokálnych sieťových súradníc je možné odhadovať vzdialenosť medzi ktoroukoľvek dvojicou uzlov. Ak dva uzly patria do rovnakej množiny, tak sa vzdialenosť medzi nimi odhaduje z lokálnych súradníc. Inak, ak dva uzly patria do dvoch rôznych množín, tak sa vzdialenosť medzi nimi odhaduje z globálnych súradníc. Tento hierarchický prístup zlepšuje presnosť odhadu vzdialenosti.

(2.1)

$$D^E(A,B) = \begin{cases} \|x_{A.local} - x_{B.local}\| & C_A = C_B \\ \left\|x_{A.global} - x_{B.global}\right\| & C_A \neq C_B \end{cases}$$

Kde c_a a c_b sú množiny, do ktorých patria uzly A a B.

Presnosť odhadu latencie pre vybraný systém sieťových súradníc je často označovaný ako relatívna chyba (RE – relative error) odhadovanej vzdialenosti ku reálnej latencii meranej v Internete. Relatívna chyba medzi uzlom i a j je definovaná ako:

(2.2)

$$RE = \frac{|L^E(i,j) - L(i,j)|}{L(i,j)}$$

Menšie hodnoty relatívnej chyby indikujú vyššiu presnosť predikcie. Ak hodnota odhadovanej latencie dosiahne hodnotu zmeranej latencie, tak sa relatívna chyba rovná nule.

Algoritmus 4: Pharos

```
Connect_to_Rendezvous_point(rp);
Get_Anchors_List(rp);
Nearest_Anchor_Distance = ∞;
```

```
foreach i in Anchors do
        d(i) = measure distance to i;
        if Nearest_Anchor_Distance > d(i) then
                Nearest_Anchor_Distance = d(i);
                Nearest_Anchor = i;
        end
end
Join_Cluster(Nearest_Anchor);
while forever do
        j = random (local neighbors of i);
        x_i.local = vivaldi (rtt, x_j.local, e_j.local);
        j = random (global neighbors of i);
        x_i.global = vivaldi (rtt, x_j.global, e_j.global);
        wait (update_interval);
end
```

2.2 Systém Myth

V algoritme Vivaldi potrebujú uzly niekoľko krokov na to, aby sa dostali do ideálnej pozície kde je energia celého pružinového systému najnižšia. Každá aktualizácia pozície uzlu sa snaží minimalizovať energiu systému priblížením sa k zmeranej vzdialenosti k niektorým zo svojich susedov. Z uvedeného vyplýva, že doba konvergencie dosahuje niekedy aj desiatky sekúnd aj v prípade, že uzly v systéme majú stabilnú polohu. Navyše je veľmi pravdepodobné, že táto doba bude narušená pripájaním a odpájaním sa susedných uzlov, čo je bežné v každom P2P systéme. Podľa štúdie [12] bolo dokázané, že skoro každý systém sa zaoberá zmenami spôsobenými častým pripájaním, odpájaním a výpadkami uzlov. Navyše, na novo pripojené uzly s ešte nepresnými súradnicami sa budú odkazovať uzly susedné s nádejou spresnenia vlastných súradníc, čo zvyšuje nestabilitu systému. Tým pádom, dlhá doba konvergencie spolu s nepresnými počiatočnými polohami uzlov môžu značne zhoršiť presnosť predikcie algoritmu Vivaldi v prostredí s vysokým náhodným odpojovaním uzlov. Pre tento prípad bol navrhnutý algoritmus Myth [5]. Jeho použitím sa inicializačné súradnice uzlov blížia svojej ideálnej polohe, tým pádom doba konvergencie je značne znížená. A navyše, keďže počiatočné súradnice uzlu v systéme Myth sú omnoho presnejšie ako tie pri použití algoritmu Vivaldi, je zredukované znehodnotenie súradníc pri odkazovaní sa na novo pripojený uzol. Experimentálne

výsledky ukázali, že v prípade prostredia s vysokým náhodným odpájaním uzlov Myth výkonovo prekonáva algoritmus Vivaldi. Ďalej v oboch prípadoch prostredia stabilného a prostredia s vysokým náhodným odpájaním uzlov dosahuje Myth skoro zhodnú presnosť predikcie.

Použitie rovnakých počiatočných súradníc pre všetky uzly, ako v algoritme Vivaldi, nie je vhodný postup pri zavádzaní systému aj keď uzly budú rozptýlené v nasledujúcich krokoch. Tým pádom pod vplyvom častého odpojovania je veľká časť uzlov v stave zmätku s nepresnými súradnicami. Na vyriešenie tohto problému systém Myth zavádza každý uzol s počiatočnými súradnicami blízke ideálnym. Dosahuje to zavedením predikčnej schémy pred použitím algoritmu Vivaldi. Táto schéma využíva súradnice uzlov, ktoré sú už zapojené do systému. Výsledkom je, že každý uzol potrebuje menej krokov aby dosiahol stav konvergencie. Zároveň je redukovaná predikčná chyba spôsobovaná odkazovaním sa na novo pripojené uzly čím je zvýšená stabilita systému. Pridaním predikčnej schémy vytvára systém Myth dodatočnú komunikačnú a výpočtovú réžiu pri pripojení uzlu. Výpočtová réžia je však zanedbateľná lebo čas potrebný na výpočet súradníc je omnoho kratší ako interval medzi aktualizačnými krokmi algoritmu Vivaldi, čo je rádovo niekoľko sekúnd.

Keď sa uzol A pripája do vrstvového systému, vytvára susedské väzby s uzlami už zapojenými vo vrstve pomocou protokolu Gossip [11]. Po získaní dostatočného množstva susedov zmeria uzol *A* latencie ku *L* uzlom vo svojej množine susedov ($L > N$, kde N udáva rozmer priestoru). Za použitia *L* zmeraných vzdialeností dokáže uzol *A* vypočítať vlastné inicializačné súradnice IC_a, ktoré minimalizujú celkovú chybu medzi zmeranými a vypočítanými vzdialenosťami. Rovnako ako v [7], sa na minimalizovanie chybovej funkcie používa Nelder-Meadova metóda. Je zjavné, že myšlienka tohto predikčného algoritmu je podobná s výpočtom súradníc u bežných uzlov v algoritme GNP. Avšak, je tu značný rozdiel medzi systémami Myth, GNP a inými LBA[3]: Myth nepotrebuje rozmiestnenie špeciálnych orientačných bodov. Trikom je využiť existujúce uzly, ktoré sú už zapojené do systému, ako orientačné body. Toto riešenie je škálovateľné a bez jediného slabého miesta, pretože tu neexistujú žiadne orientačné body.

Algoritmus 5: Myth

```
Connect_to_Rendezvous_Point(rp);
Get_Neighbor_Candidates_List(rp);
Join_Overlay();
```

[3] LBA (Landmark-Based Algorithms) – algoritmy postavené na princípe využitia orientačných bodov

```
Make_Connections_to_Neighbors();
foreach i in Neighbors do
        d(i) = Measure Distance to i;
end
x_i = Initial_Coordinates_Prediction(d(.),NCs(.));
Wait (Update_Interval);
while forever do
        J = random (neighbors of i);
        x_i = vivaldi (rtt, x_j, e_j);
        Wait (Update_Interval);
end
```

2.3 Systém PCoord

PCoord [10] je plne decentralizovaný sieťový pozičný systém, kde každý uzol iteratívne aktualizuje svoje súradnice aby vyladil presnosť predikcie odhadovaných pozícií. Každý uzol aktualizuje svoje súradnice za účelom minimalizácie stratovej funkcie ktorá meria rozdiel medzi aktuálnou a geometrickou vzdialenosťou medzi sebou a malým súborom ostatných uzlov. Algoritmus PCoord obsahuje nasledovné mechanizmy:

- Váženú stratovú funkciu ktorá dovoľuje vzorkovaným súradniciam s vyššou presnosťou predikcie mať vyššiu váhu v stratovej funkcii.
- Vážený faktor odolnosti v stratovej funkcii, ktorý pomáha stabilizovať proces konvergencie.
- Prahovací mechanizmus na zmenšenie počtu uzlov pohybujúcich sa k novým súradniciam pomocou faktoru, ktorý je nepriamo úmerný chybe dávky vzorkovaných súradníc a vzdialeností.

Vážená stratová funkcia zabraňuje uzlom aby reagovali príliš rýchlo na zlé referenčné body a napomáha rozoznať medzi uzlami s rozdielnou presnosťou predikcie súradníc. Dosahuje to váhovaním straty, ktorou každý referenčný uzol prispieva k predikcii podľa ich súradníc. Váha je vypočítaná podľa presnosti relatívnej predikcie každého referenčného bodu tak, aby uzly s presnejšími súradnicami mali väčší vplyv na výsledok ako menej presnejšie uzly.

Vážený faktor odolnosti v stratovej funkcii zavádza mechanizmus na redukovanie oscilácie uzlov. To sa deje, keď uzly aktualizujú svoje pozície len na základe meraní ku aktuálnej dávke orientačných bodov a súradnice vytvorené z predošlých vzoriek nemajú žiadny vplyv na výsledok iterácie. Mechanizmus pracuje na jednoduchom princípe použitia aktuálnych súradníc uzlu ako faktor odolnosti na výpočet nových súradníc uzlu i. Kedykoľvek uzol i počíta svoje nové súradnice c_i^{new}, pridá sám seba ako *(M + 1)-tý* uzol do svojho zoznamu referenčných bodov. Tým pádom sa aktuálne súradnice c_i stávajú faktorom odolnosti v stratovej funkcii, ktorý penalizuje posun uzlu na novú pozíciu. Ďalej je to váhované relatívnou presnosťou predikcie (relatívnou k ostatným referenčným uzlom k *i*) súradníc uzlu i tak, že čím si je uzol istejší o presnosti svojich súradníc, tým silnejší je faktor odolnosti. Presnejšie popísanie váženej stratovej funkcie s faktorom odolnosti je nasledovné:

(2.3)

$$\varepsilon = w_i(d_{ii} - \|c_i^{new} - c_i\|)^2 + \sum_{j=1}^{j=M} w_j\left(d_{ij} - \left\|c_i^{new} - c_j\right\|\right)^2$$

Vážená stratová funkcia popísaná vyššie pomáha redukovať negatívny efekt referenčných bodov s vysokou chybou predikcie. Avšak, chyba predikcia nie vždy dostatočne popisuje, či vzdialenosť medzi uzlami i a j slúži ako dobrá vzorka na odhad týchto dvoch uzlov.

Pre tento prípad zavádza algoritmus PCoord mechanizmus, ktorý dovoľuje uzlu upraviť svoje súradnice na základe určitej dávky vzoriek podľa indexu „vhodnosti“. Myšlienkou je, že dávka vzoriek obsahujúca nevhodné vzdialenosti bude mať vyššiu zvyškovú chybu ako dobrá dávka vzoriek. Aby sa predišlo reakcii na dávku s nevhodnými vzorkami, udržuje si uzol v systéme PCoord plávajúci vážený priemer chyby vhodnosti. Uzol následne priraďuje váhu jednotlivým dávkam vzoriek ako funkciu pomeru medzi priemernou a aktuálnou chybou vhodnosti a potom sa rozhodne ako má reagovať na aktuálnu dávku vzoriek podľa vypočítanej váhy. Tým pádom, ako chyba vhodnosti aktuálnej dávky presiahne priemernú chybu vhodnosti, tak uzol zmenší veľkosť posunutia smerom k novým súradniciam podľa faktoru ρ. Je to pomer medzi priemernou a aktuálnou chybou vhodnosti. Nižšie je uvedený pseudokód algoritmu, ktorý sumarizuje všetky podmienky:

Algoritmus 6: PCoord

PCoord() {

$c_i = c_{origin}$

while *(in the system) {*

Samples = SamplePeers()

UpdatePredictionError(Samples)

// pridaj súradnice lokálneho uzlu i medzi vzorky

Samples = Samples.add(i)

(c_i^{new}, c_i^{newf}) = MinimizeWeightedError(Samples, c_i)

$c_i = c_i^{new}$

UpdateFitError(e_i^{newf})

ProbeNearNeighbors() } }

MinimizeWeightedError (Samples, c_{guess}) {

$e_{TOP}^{p} = 1 + \tau$

foreach *node k in samples {*

// priraď váhu každému uzlu zo vzoriek

$a_k = e_{TOP}^{p} - e_k^{p}$

$w_k = \frac{a_k^2}{\sum_{j \in Samples} a_j^2}$

} // ukonči for

// teraz nájdi nové súradnice

// Nájdi také c_i^{new} čo minimalizuje

$\sum_{k \in Samples} w_k (d_{ik} - \| c_i^{new} - c_k \|)^2$

// e_i^{newf} is zvyšková chyba po minimalizácii

$e_i^{newf} = \sum_{k \in Samples} w_k (d_{ik} - \| c_i^{new} - c_k \|)^2$

// posunutie k novým súradniciam

$\rho = MIN\left(\frac{e_i^f}{e_i^{newf}}, 1\right)$

// posun o ρ kus k novému riešeniu

$c_i^{new} = c_i + \left(\rho * (c_i^{new} - c_i)\right)$

return $\left(c_i^{new}, e_i^{newf}\right)$

} //ukonči MinimizeWeightedError

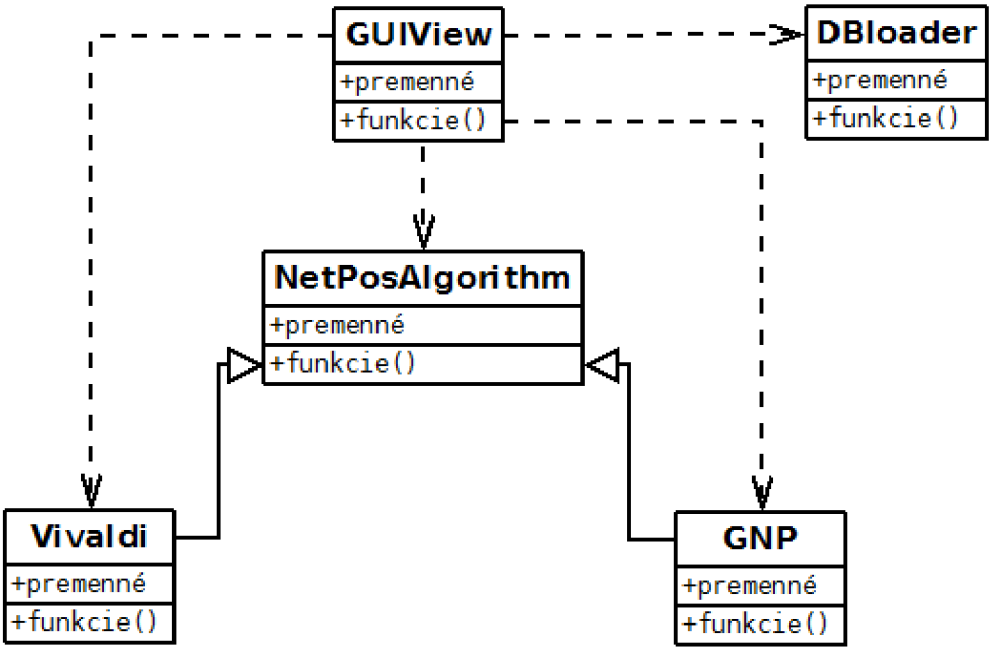
3 VÝVOJ SIMULAČNEJ KNIŽNICE

Ako programovací jazyk pre vývoj simulačnej knižnice schopnej pracovať s databázou odoziev RTT bola vybraná Java. Java je moderný objektový programovací jazyk a bol zvolený hlavne preto, že poskytuje širokú škálu nástrojov a prostriedkov pre grafický výstup simulačného prostredia. Taktiež je nezávislý na použitej platforme (t.j., programy vytvorené v tomto jazyku sa dajú spustiť na ľubovoľnom operačnom systéme) za predpokladu prítomnosti JVM (Java Virtual Machine) – kompilátoru Java programov.

Základom pre aplikáciu je aplikačný rámec (framework) Swing Application Framework, ktorý je založený na knižniciach grafického užívateľského rozhrania Swing. K behu aplikácie je potrebné mať nainštalované prostredie JRE 1.6 (Java Runtime Environment) alebo JDK 1.6 (Java Development Kit). Na vývoj simulačnej knižnice bolo použité vývojové prostredie NetBeans z dôvodu pokročilej implementácie nástrojov pre tvorbu grafického rozhrania.

3.1 Architektúra knižnice

V tejto diplomovej práci je momentálne riešená implementácia algoritmov v 2-rozmernom priestore. Pre účely použitia tejto knižnice len na dáta zmerané v sieti PlanetLab je vybraný programovací jazyk veľmi vhodný. Knižnica z dôvodu jednoduchosti a názornosti bola navrhnutá tak, že implementácie jednotlivých algoritmov sú rozdelené len do dvoch tried, pričom ďalšie triedy slúžia na definovanie užívateľského rozhrania a popis funkcií na prácu s dátovými súbormi. Schéma celej knižnice je zobrazená na obrázku obr. 5.1.



Obr. 3.1: UML diagram vyvinutej knižnice

Celú knižnicu tvoria štyri triedy: DBLoader, Vivaldi, GNP a NetPosAlgorithm. Jej základom sú však triedy Vivaldi a GNP, ktoré implementujú samotné algoritmy. Obidve tieto triedy dedia vlastnosti rodičovskej triedy NetPosAlgorithm, ktorá spája v sebe metódy a premenné na prácu so súradnicami a latenciami jednotlivých uzlov. Trieda DBLoader obsahuje metódy potrebné na prácu s dátami v súboroch. Metódy z tejto triedy sú volané vždy pri otváraní nového dátového súboru. Nadstavbový balík s grafickým užívateľským rozhraním následne pri interakcii s užívateľom volá metódy na prácu so súbormi, na výpočet pozícií uzlov ako aj na export dát do súboru.

3.2 Trieda NetPosAlgorithm a popis vybraných funkcií

Trieda NetPosAlgorithm je rodičovskou triedou pre triedy Vivaldi a GNP. Definuje premenné a funkcie používané obidvoma potomkami. Zoznam premenných je zobrazený v tabuľke 3.1.

Tab. 3.1: Zoznam premenných triedy NetPosAlgorithm

Názov premennej	Typ	Význam premennej
nodesList	ArrayList<String>	Zoznam uzlov z dátového súboru

nodesDistances	Hashtable<String, Float>	Zoznam väzieb a k nim priradených RTT
nodesCoordinatesComputed	Hashtable<String, Point2D>	Zoznam uzlov a ich vypočítané súradnice
nodeErrors	Hashtable<String, Float>	Zoznam uzlov a ich chyba
aeHistory	ArrayList<Double>	Pole absolútnych chýb celého systému
dreHistory	ArrayList<Double>	Pole smerových relatívnych chýb celého systému
reHistory	ArrayList<Double>	Pole relatívnych chýb celého systému

- public double getNodesLatency(String from, String to) – funkcia vracia hodnotu RTT medzi dvomi uzlami.
- public void initializeNodesPosition() – funkcia initializeNodesPosition generuje náhodné súradnice uzlov v intervale <-50;50>
- public double computeDistance(Point2D p1, Point2D p2) – funkcia computeDistance na základe vstupných súradníc dvoch bodov vypočíta vzdialenosť medzi nimi pomocou Pytagorovej vety
- public Hashtable<String, Double> getSystemErrors(String node, ArrayList<String> nodes) – funkcie getSystemErrors vracia hashtable o troch chybách: absolútnej, relatívnej a DRE.

3.3 Trieda Vivaldi a popis vybraných funkcií

Táto trieda dedí vlastnosti triedy NetPosAlgorithm a implementuje verziu algoritmu Vivaldi s adaptívnym časovým krokom. Obsahuje premenné a metódy potrebné na výpočet jednotlivých iterácii. Jedna iterácia predstavuje výpočet nových súradníc pre každý uzol zo zoznamu uzlov. Zoznam premenných tejto triedy je vymenovaný v tabuľke:

Tab. 3.2: Zoznam premenných triedy Vivaldi

Názov premennej	Typ	Význam premennej
ce	Double	Hodnota ladiaceho parametru ce

cc	Double	Hodnota ladiaceho parametru cc
lastError	Double	Záznam o poslednej chybe
semiLastError	Double	Záznam o predposlednej chybe
errorHistory	ArrayList<Double>	Udržuje záznam o všetkých chybách

- public void VivaldiAdaptiveTimestep() – funkcia VivaldiAdaptiveTimestep implementuje samotný algoritmus s adaptívnym časovým krokom. Taktiež vytvára inštancie a napĺňa polia troch popisných chýb ktorých výpočet sa vykonáva taktiež v tejto funkcii.
- public void runVivaldiUntil() – funkcia runVivaldiUntil volá funkciu VivaldiAdaptiveTimestep a definuje podmienky koľkokrát má byť volaná. V mnou vykonávaných simuláciách bol volaný algoritmus Vivaldi dovtedy, kým rozdiel medzi poslednou a predposlednou chybou nebol menší alebo rovný 0.00001.

3.4 Trieda GNP a popis vybraných funkcií

Táto trieda taktiež dedí vlastnosti triedy NetPosAlgorithm a implementuje Nelder-Meadovu metódu. Obsahuje premenné a metódy potrebné na výpočet pozícií všetkých uzlov. Jedna iterácia taktiež predstavuje výpočet nových súradníc každého uzlu zo zoznamu orientačných bodov alebo zoznamu staníc. Súhrn premenných tejto triedy je možné vidieť v tabuľke:

Tab. 3.3: Zoznam premenných triedy GNP

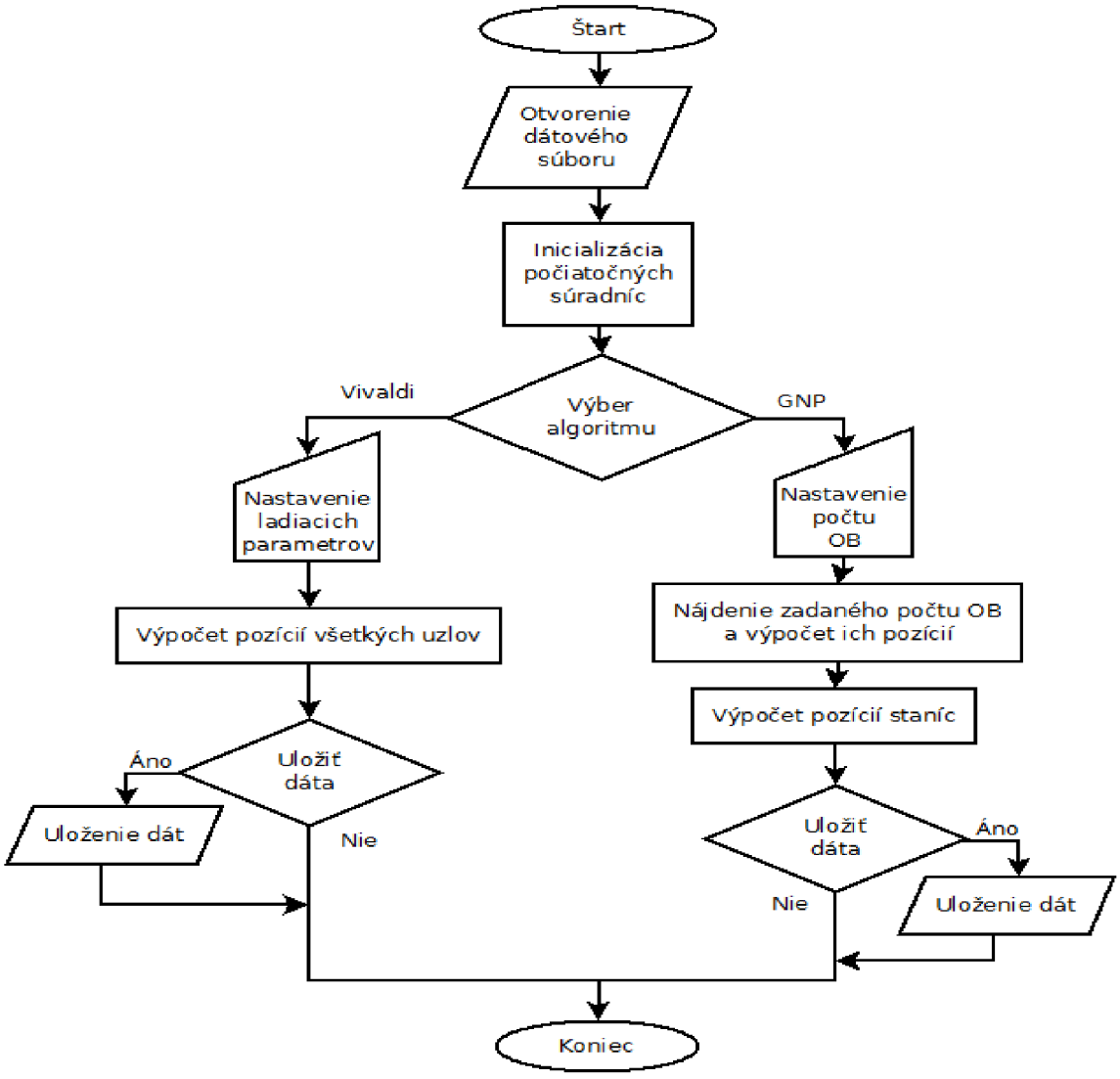
Názov premennej	Typ	Význam premennej
numberOfLandmarks	int	Udržuje počet orientačných bodov zadaný užívateľom
lastLandmarkError	double	Udržuje chybu orientačných bodov po vykonaní jednej iterácie
semiLastLandmarkError	double	Udržuje predposlednú chybu orientačných bodov po vykonaní jednej iterácie
lastHostError	double	Udržuje chybu staníc po vykonaní jednej iterácie

semiLastHostError	double	Udržuje predposlednú chybu staníc po vykonaní jednej iterácie
landmarkList	ArrayList<String>	Udržuje zoznam orientačných bodov
hostList	ArrayList<String>	Udržuje zoznam staníc
landmarkSimplex	Hashtable<String, Hashtable<String, Point2D>>	Udržuje zoznam orientačných bodov a súradnice k nim priradených vrcholov simplexu
hostSimplex	Hashtable<String, Hashtable<String, Point2D>>	Udržuje zoznam staníc a súradnice k nim priradených vrcholov simplexu
landmarkErrorHistory	ArrayList<Double>	Udržuje záznam o chybách

- public ArrayList<String> getFriendlyNodes(String node1, ArrayList<String> nodes) – funkcia getFriendlyNodes vyhľadáva všetky uzly so zmeranou latenciou k uzlu node1 z poľa nodes.
- public ArrayList<String> findNlandmarks(int n, String landmark, ArrayList<String> landmarkSet) – rekurzívna funkcia findNlandmarks vyhľadáva také uzly, ktoré majú záznam o latencii ku všetkým uzlom nájdeným funkciou getFriendlyNodes. Následne volá samú seba znova nájde všetky uzly so zmeranou latenciou k doteraz nájdeným OB.
- public ArrayList<String> findGNPLandmarks() – funkcia findGNPlandmarks volaním funkcie findNlandmarks nájde prvú N-ticu orientačných bodov, ktorých počet N je zadaný užívateľom cez grafické rozhranie.
- public Hashtable<String, Point2D> generateSimplexCoordinates() – funkcia generateSimplexCoordinates generuje náhodné súradnice vrcholov simplexu z intervalu <-50;50>. Simplex je pomenovaný OPQ.
- Hashtable<String, Double> simplexVerticesError(String landmark, Hashtable<String, Point2D> simplexVerticesCoords, ArrayList<String> landmarks) – funkcia simplexVerticesError vracia hodnoty chýb jednotlivých vrcholov simplexu.
- public ArrayList<String> sortVerticesByError(Hashtable<String, Double> simplexVerticesErrors) – funkcia sortVerticesByError vzostupne zoraďuje vrcholy vstupného simplexu podľa ich chýb.

- public Double pointError(String landmark, Point2D pointCoords, ArrayList<String> landmarks) – funkcia pointError vracia chybu bodu voči orientačným bodom. Využíva sa pri výpočte minima bodov R, E, M, S, C v Nelder-Meadovej metóde.
- public void simplexDownhillAlgorithm() – funkcia simpleDownhillAlgorithm implementuje samotný algoritmus. Ďalej počíta a ukladá vývoj popisných chýb.
- public void simplexDownhillAlgorithmForHosts() – funkcia simplexDownhillAlgorithmForHosts tiež implementuje samotný algoritmus ale je volaná len pre nájdenie pozícií staníc.

Činnosť tried Vivaldi a GNP je popísaná vývojovým diagramom na obr. 3.2.



Obr. 3.2: Životný cyklus objektov tried Vivaldi a GNP

3.5 Trieda DBLoader

V tejto triede dochádza k načítaniu databáze s nameranými hodnotami zo siete PlanetLab. Slúži ako predspracovanie dát pred použitím triedami implementujúcimi jednotlivé algoritmy. Zoznam použitých premenných je v nasledujúcej tabuľke.

Tab. 3.4: Zoznam premenných triedy DBLoader

Názov premennej	Typ	Význam premennej
filename	String	Udržuje cestu k súboru s dátami
nodesList	ArrayList<String>	Udržuje zoznam uzlov z dátového súboru
nodesDistancesVector	Hashtable<String, Vector<Float>>	Udržuje zoznam väzieb a ich parametre: RTT, hops a vzdialenosť

- public void loadDB() throws NumberFormatException, IOException – funkcia loadDB prechádza jednotlivé riadky súboru s dátami a hodnoty ukladá do premenných z tabuľky.

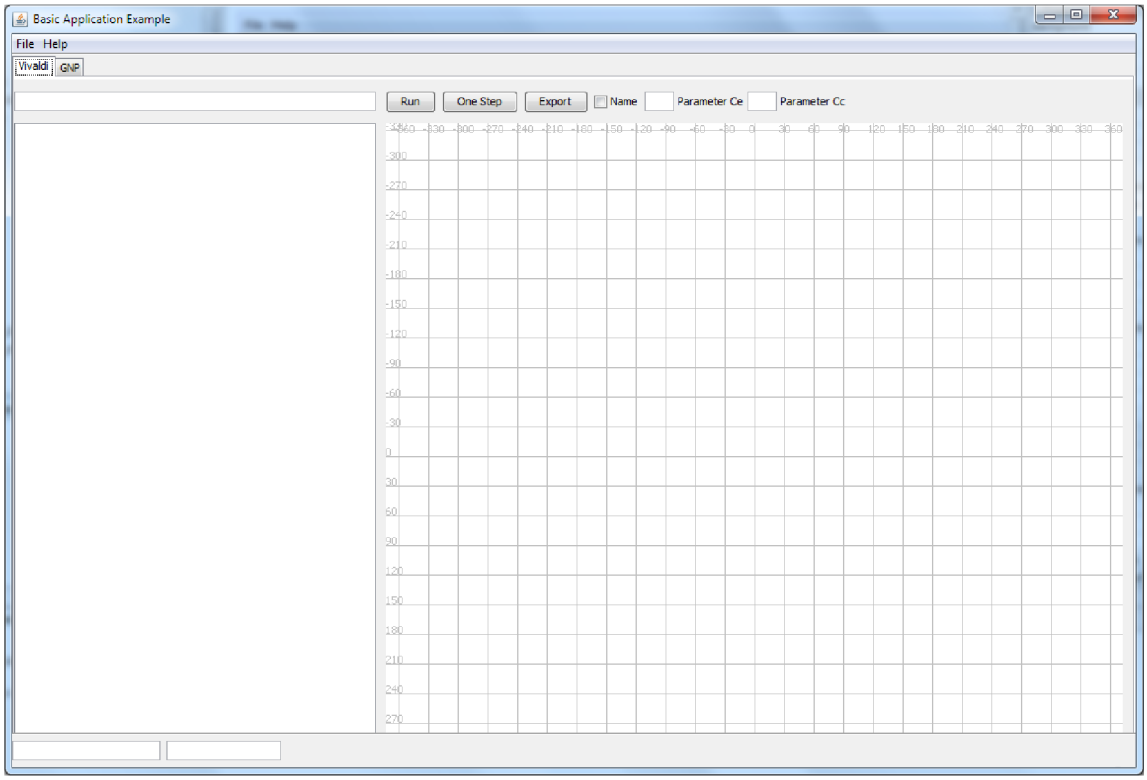
3.6 Popis rozhrania

Rozhranie aplikácie je jednoduché ovládateľné z hlavného okna. Toto hlavné okno obsahuje dve karty, ktoré efektívne sprístupňujú väčšinu funkcií. Prvá karta Vivaldi slúži na simulovanie pohybu uzlov pomocou algoritmu Vivaldi a zároveň ich zobrazenie polohy a súradníc. Ďalšia karta GNP obsahuje funkcie na simulovanie pohybu orientačných bodov a staníc, a zobrazenie ich súradníc. Hlavné okno je doplnené lištou ponúk, kde môže užívateľ cez ponuku *File* otvoriť súbor s nameranými dátami.

3.6.1 Karta Vivaldi

Priebeh simulácie pohybu uzlov pomocou algoritmu Vivaldi prebieha na karte Vivaldi. Po otvorení dátového súboru sa v textovom poli umiestnenom nižšie k nemu ihneď zobrazí systémová cesta. Táto karta obsahuje tri tlačidlá: *Run*, *One Step* a *Export*. Tlačidlo *Run* slúži na spustenie simulácie, ktorá vykonáva jednotlivé iterácie algoritmu dovtedy, kým nie sú splnené implicitne nastavené podmienky. Tlačidlo *One Step* vykoná

jednu iteráciu algoritmu. Posledné tlačidlo *Export* ukladá priebeh chýb do súboru. Ďalej táto karta obsahuje zaškrtávací box, po ktorého zaškrtnutí dôjde k zobrazeniu názvu uzlov v mriežke. Ďalej táto karta obsahuje textové polia na zadanie hodnoty ladiacich parametrov c_e a c_c. Tieto parametre je potrebné zadať pred samotným spustením simulácie, inak dôjde k zobrazeniu dialógového okna. Hlavným prvkom tejto karty je mriežka, v ktorej dochádza k vykresleniu jednotlivých pozícií uzlov. Naľavo od mriežky je textové pole kde dochádza k výpisu súradníc uzlov po aktuálnej iterácii algoritmu.



Obr. 3.3: Grafické užívateľské rozhranie aplikácie

3.6.2 Karta GNP

Karta GNP sa s kartou Vivaldi líši jedine v ovládacích prvkoch a zobrazení uzlov v mriežke. Ovládacích tlačidiel simulácie je, naproti karte s algoritmom Vivaldi, päť. Prvé, *Landmark Run Until*, simuluje proces nájdenia zadaného počtu orientačných bodov dovtedy, kým nie sú splnené implicitne zadané podmienky. Tlačidlo *Landmark One Step* vykoná jednu iteráciu Nelder-Meadovho algoritmu. Po nájdení vhodnej pozície orientačných bodov s minimálnou chybou predikcie je potrebné v ďalšom kroku nájsť

vhodnú pozíciu staníc. K tomu slúžia tlačidlá *Host Run Until*, ktoré simuluje proces lokalizácie staníc pokým nie je splnená implicitne nastavená podmienka, a tlačidlo *Host One Step* ktoré vykoná jednu iteráciu algoritmu. Funkcie tlačidla *Export* a zaškrtávacieho boxu *Name* sú rovnaké ako v karte Vivaldi. Pred samotným spustením lokalizácie je potrebné zadať počet orientačných bodov do textového poľa s označením *Počet landmarkov* inak dôjde k zobrazeniu dialógového okna.

Obr. 3.4: Ovládacie prvky karty GNP

3.6.3 Popis vstupného súboru

V tejto diplomovej práci bolo mojou úlohou pracovať so súborom nameraných dát zo siete PlanetLab. Bolo potrebné definovať syntax vstupných dát, pre prípad rozšírenia knižnice o ďalšie algoritmy treťou stranou. Syntax má nasledovný tvar

názovuzlu1;názovuzlu2;hodnotartt;počethopov;vzdialenosť

Algoritmy Vivaldi a GNP podľa definícií pracujú len s hodnotou RTT a tým pádom aj mnou vyvinutá knižnica pracuje len s touto položkou. Reálny záznam s databáze vyzerá nasledovne:

planetlab1.unineuchatel.ch;146-179.surfsnel.dsl.internl.net;23.400;15;642.968

3.7 Testy knižnice

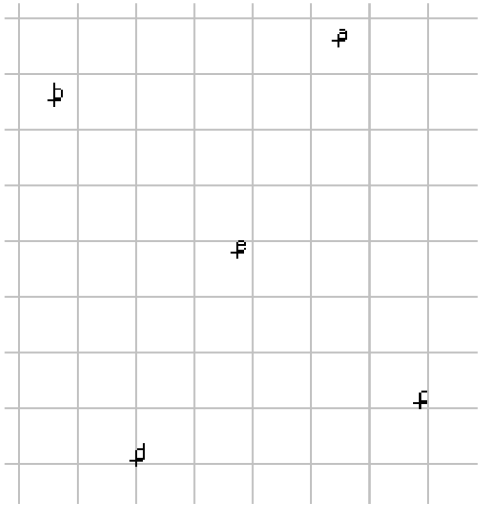
Na overenie funkčnosti a konvergencie obidvoch algoritmov boli vykonané rôzne testy. Jednalo sa o jednoduché simulácie s malým počtom uzlov. Ich cieľom bolo zistenie základného chovania implementácie algoritmov. Výsledky testov pre všetky konfigurácie sietí sú prezentované grafmi celkovej chyby systému a výslednou polohou uzlov.

3.7.1 Konvergencia siete s tromi uzlami

Prvým a najjednoduchším testom bola simulácia pohybu troch uzlov, ktoré vo výslednej polohe mali tvoriť trojuholník s násobkami strán Pytagorovho trojuholníka. V dátovom súbore pre tento test som nadefinoval hodnoty latencií medzi jednotlivými uzlami. Pre obidva algoritmy došlo k ustáleniu polohy uzlov tak, že vo výslednej polohe tvorili naozaj pravouhlý trojuholník. Vzhľadom k tomu, že pri každej novej simulácii bolo prvotné rozloženie uzlov rozdielne, z dôvodu náhodného priradenia súradníc v prvom kroku algoritmov, bola výsledná poloha taktiež vždy rozdielna. Avšak vzájomné vzdialenosti zodpovedali nadefinovaným latenciám. Počiatočné pozície mali vplyv len na dobu konvergencie.

3.7.2 Konvergencia siete s piatimi uzlami

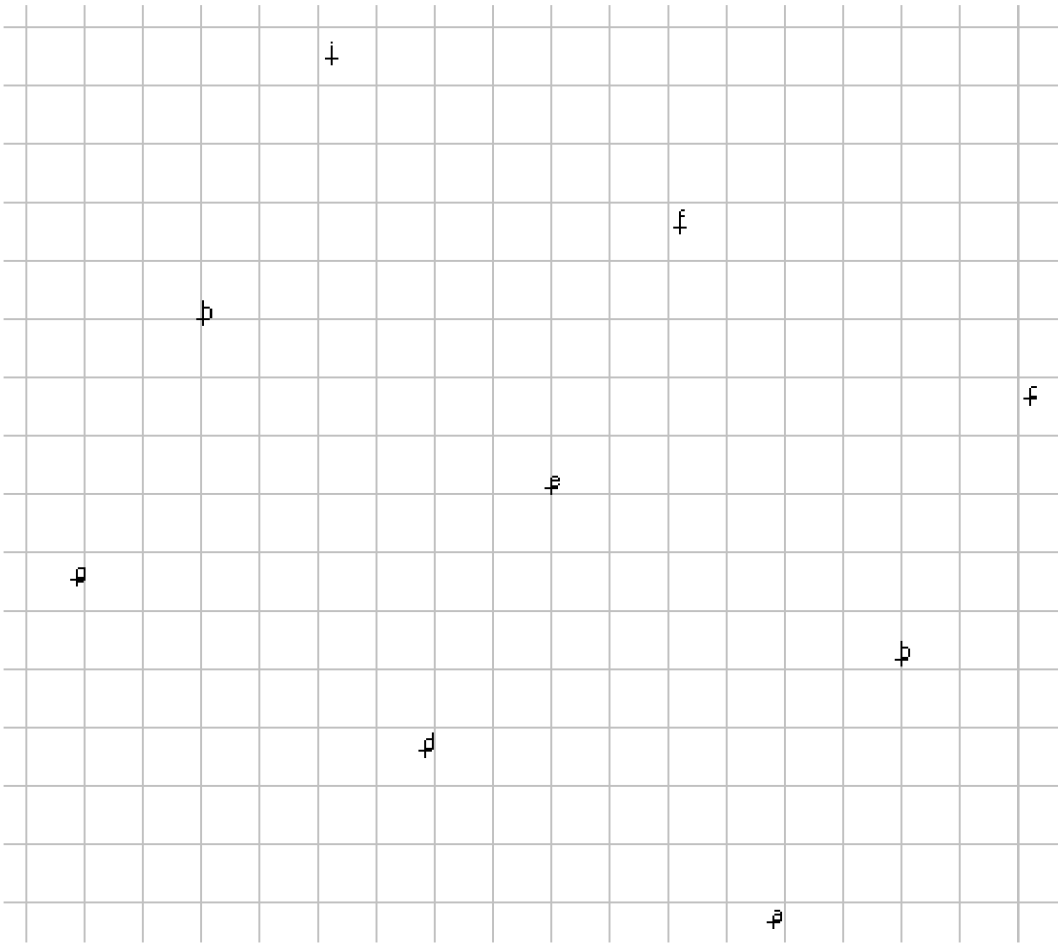
Ďalším testom bola konvergencia modelu siete o piatich uzloch. Uzly mali medzi sebou nadefinované latencie tak, že tvorili obdĺžnik s pomerom dĺžok strán ako Pytagorov trojuholník, pričom piaty uzol ležal na priesečníku uhlopriečok. Všetky body mali prvotnú pozíciu náhodne vygenerovanú v intervale <-50;50> a tak výsledná poloha bola vždy rozdielna. Avšak vzdialenosti medzi uzlami zodpovedali nadefinovaným latenciám. Obidva algoritmy sa k poslednej iterácii, kde rozdiel medzi poslednou a predposlednou chybou bol zanedbateľne malý, dostali v rôznom počte krokov. Pre ilustráciu je výsledná poloha uzlov u algoritmu Vivaldi zobrazené na obr. 5.5.



Obr. 3.5: Výsledná poloha piatich uzlov u algoritmu Vivaldi

3.7.3 Konvergencia siete s deviatimi uzlami

Posledným testom bola konvergencia modelu siete o deviatich uzloch. Každý mal nadefinovanú latenciu k šiestim ďalším a to tak, že ak bol uzol vrcholom modelu siete, nepoznal latenciu k uzlom na stredoch dvoch protiľahlých strán, a opačne. Obidva algoritmy v 80% prípadov dospeli v poslednej iterácii k správnej polohe uzlov. Z dôvodu nie 100% úspešnosti som predpokladal, že mnou vyvinutá implementácia je chybná. Dospel som však k záveru, že v závislosti na rozložení uzlov pri inicializácii náhodných súradníc algoritmy skonvergujú k lokálnemu minimu. Výsledná poloha uzlov je zobrazená na obr. 5.6.



Obr. 3.6: Výsledná poloha deviatich uzlov u algoritmu Vivaldi

4 NAMERANÉ SIMULÁCIE

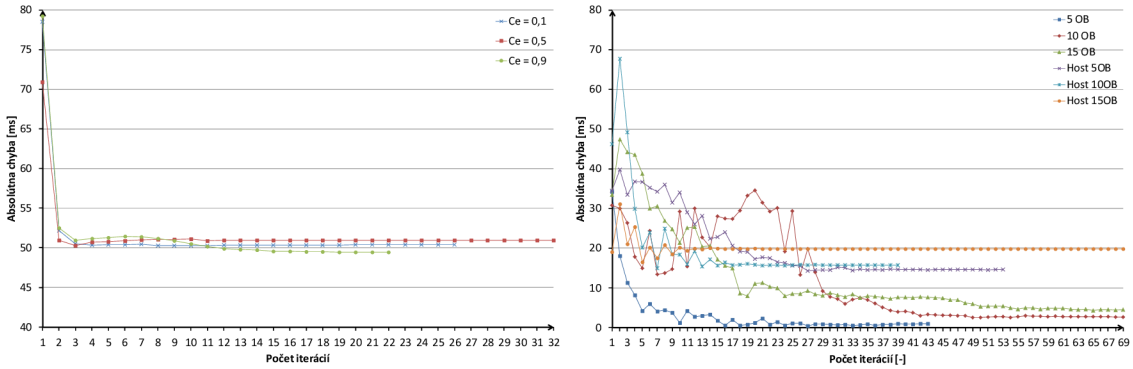
V rámci semestrálnej práce boli vykonané nedokumentované simulácie na dátach zo siete PlanetLab pomocou centralizovanej verzie algoritmu Vivaldi. V rámci diplomovej práce bol algoritmus prepracovaný na verziu s adaptívnym časovým krokom a bolo vyhodnotené jeho chovanie v simuláciách a porovnané s Nelder-Meadovým algoritmom systému GNP. Súbor dát obsahoval 19209 záznamov o väzbách medzi uzlami. V celom súbore je celkovo 204 rôznych uzlov. Keďže, ako bolo popísané v kapitole 2. a 3., každý z algoritmov minimalizuje odlišnú chybu, bolo potrebné stanoviť univerzálne chyby pre zhodnotenie presnosti výpočtov. Uvažoval som so smerovou relatívnou chybou, relatívnou chybou a absolútnou chybou. Smerová relatívna chyba (*directional relative error*) vyjadruje, ako veľmi sa predpovedaná vzdialenosť zhoduje s korešpondujúcou zmeranou vzdialenosťou. Je definovaná ako:

$$DRE = \frac{P_{DIST} - M_{DIST}}{min(P_{DIST}, M_{DIST})} \tag{4.1}$$

Z uvedenej rovnice vyplýva, že nulová hodnota *DRE* zodpovedá dokonalej predikcii, hodnota rovná jednej zodpovedá dvojnásobne väčšej predpovedanej vzdialenosti a hodnota rovná mínus jednej zodpovedá polovičnej vzdialenosti. Pre popis všeobecnej presnosti algoritmov som použil relatívnu chybu, čo je v skutočnosti len absolútna hodnota z *DRE*. Ako poslednú popisnú chybu som použil absolútnu, ktorá vyjadruje o koľko milisekúnd sa predpovedaná vzdialenosť líši od vypočítanej.

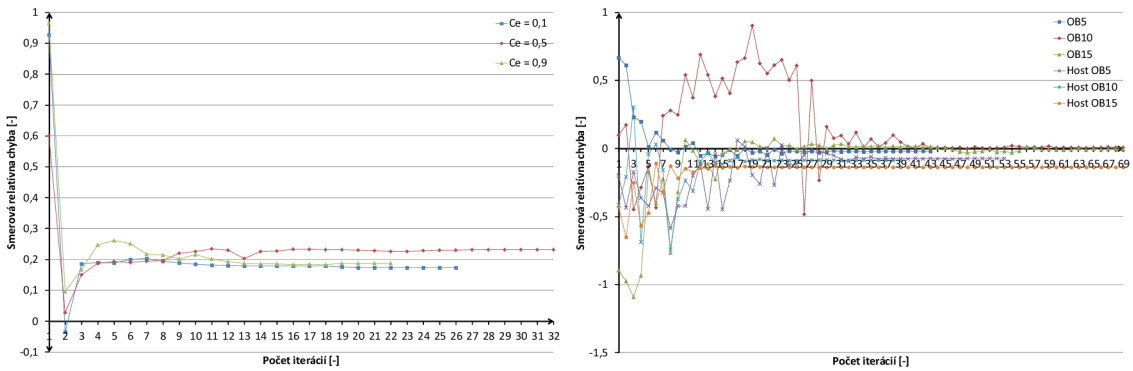
$$AE = |P_{DIST} - M_{DIST}| \tag{4.2}$$

Pre účely hlbšieho preskúmania a vyhodnotenia funkčnosti jednotlivých systémov som vykonal simulácie na piatich rôznych scenároch. Jednalo sa o simulovanie na samotnom dátovom súbore so všetkými uzlami a ďalej simulovanie na podmnožinách tohto súboru. Rozdelenie do podmnožín som vykonal na základe geografickej polohy uzlov na štyri oblasti: Severnú Ameriku, Južnú Ameriku, Európu a Áziu. V prvom scenári som simuloval chovanie systémov na všetkých uzloch. Na tomto dátovom súbore boli vykonané tri simulácie s rôznym ladiacim parametrom c_e nastaveným na hodnoty 0,9, 0,5, 0,1. Parameter c_c som podľa doporučení na reálne siete nastavil na hodnotu 0,25. Výsledky simulácii sú zobrazené v grafoch.

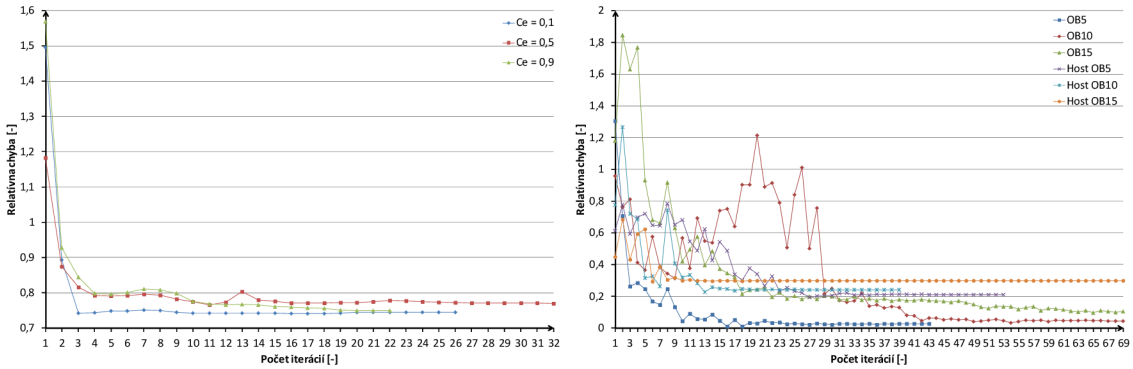


Obr. 4.1: Priebeh absolútnej chyby pre: a) algoritmus Vivaldi, b) systém GNP

Ako je z grafov možné vidieť, hodnota absolútnej chyby sa pri začiatku simulácie drží na 70 ms pre všetky tri nastavenia ladiaceho parametru c_e. Následne sa ustáli na hodnote 50 ms. To znamená, že absolútna odchýlka uzlov od zmeranej hodnoty latencie je 50 ms a nastavenie parametra c_e nemá na výsledný priebeh skoro žiadny vplyv. Ďalšia meraná chyba bola smerová relatívna chyba a relatívna chyba. Ich priebeh pre tri rôzne hodnoty parametru c_e je možné vidieť na obr. 6.2.



Obr. 4.2: Priebeh smerovej relatívnej chyby pre: a) algoritmus Vivaldi, b) systém GNP



Obr. 4.3: Priebeh relatívnej chyby pre: a) algoritmus Vivaldi, b) systém GNP

Hodnota DRE sa po niekoľkých iteráciách ustáli na hodnote 0,17 a hodnota relatívnej chyby na hodnote 0,77. Tento scenár som simuloval niekoľko krát a hodnoty výsledných chýb sa vždy držali na uvedených hodnotách. Z tohto je možné usúdiť, že algoritmus naozaj našiel celkové globálne minimum.

Ako vysvetlením, prečo sú chyby relatívne vysoké, môže byť fakt, že pri simulácii v dvojrozmernom priestore dochádza k porušeniu trojuholníkovej nerovnosti. Pri použití viacrozmerného priestoru, napríklad 2-rozmerný s výškou špeciálne navrhovaný v [1] pre Vivaldi, by došlo k minimalizovaniu medziuzlových síl smerom do nového rozmeru, výšky.

Ďalej som simuloval chovanie systému GNP na rovnakom súbore dát pre tri rôzne počty orientačných bodov: 5, 10 a 15. Orientačné body spomedzi všetkých uzlov vyberá aplikácia tak, aby mal každý bod spojenie s každým – tzv. full mesh topológia. V [7] bolo dokázané, že tento systém dosahuje najlepšie hodnoty predikcie pre 5 a 7-rozmerné priestory. Tým pádom som očakával, že výsledky budú vykazovať značnú chybu predikcie.

Vo vyššie uvedených grafoch sú zobrazené chyby vždy zvlášť pre orientačné body a stanice. Je to z toho dôvodu, že mnou implementovaná Nelder-Meadova metóda optimalizuje polohu uzlov a počíta jednotlivé chyby zvlášť pre orientačné body a zvlášť pre stanice. Nebolo by vhodné chyby sčítavať, pretože by mohlo dôjsť ku skresleniu výslednej hodnoty. Zároveň ak došlo k nájdeniu najlepšej pozície orientačných bodov v odlišnom počte iterácii ako pri hľadaní pozície staníc, tak nie je možná žiadna spoločná operácia s nameranými hodnotami.

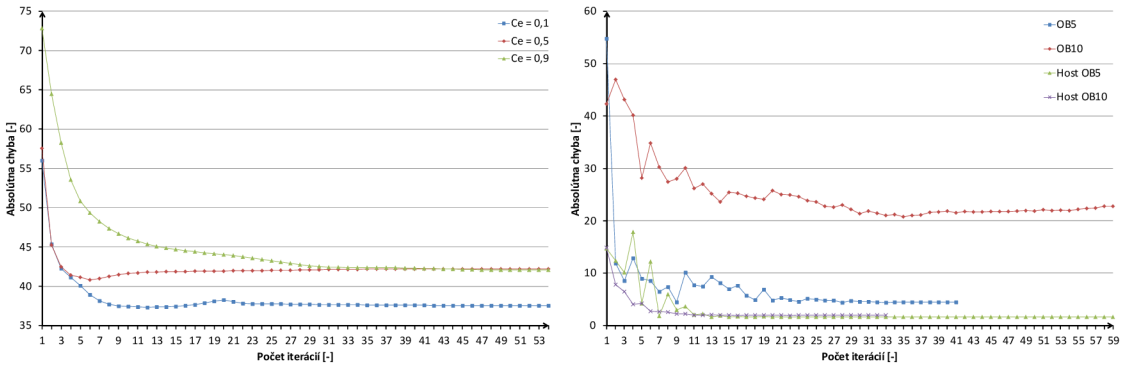
Ako je z grafov absolútnej chyby na obr. 6.1 viditeľné, pre vzostupný rast počtu orientačných bodov dosahuje hodnoty vyššie. Pre päť OB sa absolútna chyba orientačných bodov blíži k nule, pre desať k 2 ms a pre pätnásť OB sa blíži ku 4 ms. K tomu hodnota absolútnej chyby pre stanice rastie s počtom OB z 15 ms pri piatich na 20 ms pri

pätnástich. Z uvedeného vyplýva predpoklad, že s rastúcim počtom OB sa vnáša do predikcie staníc a zároveň orientačných bodov čoraz väčšia chyba.

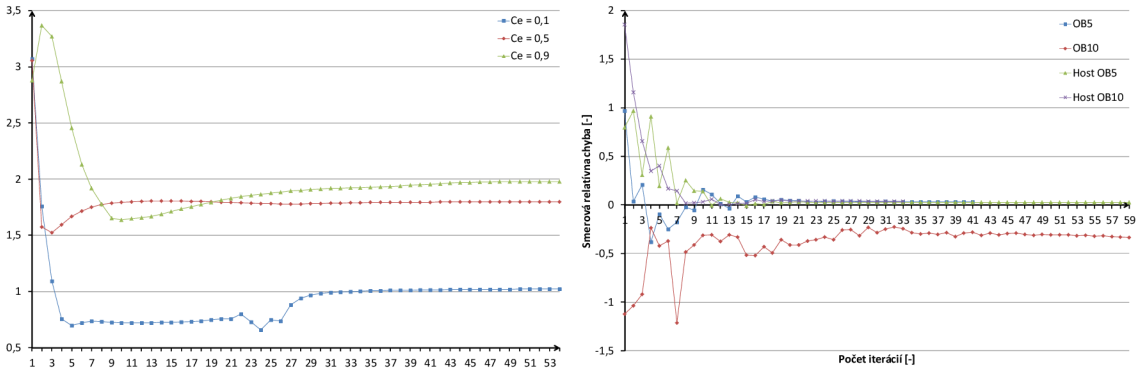
Ďalším meraným parametrom bola smerová relatívna chyba a relatívna chyba. Ich grafy priebehu je možné vidieť na obr. 6.2 a 6.3. DRE a relatívna chyba orientačných bodov pri ich rôznom počte dosahovali podobné hodnoty, tým pádom je možné tvrdiť, že počet OB nemá vplyv na ich priebeh. Avšak pri simulácii lokalizácie polohy staníc došlo k miernemu nárastu smerovej relatívnej chyby z -0,07 pri piatich OB na -0,13 pri pätnástich. Logicky došlo k nárastu relatívnej chyby z 0,21 pri piatich OB na 0,29 pri pätnástich. Tým pádom sa potvrdil predpoklad, že pre daný dátový súbor s vyšším počtom OB rastie aj chyba ktorú tieto uzly vnášajú do predikcie polohy staníc.

Z celkového pohľadu sa môže javiť, že pre daný dátový súbor je systém GNP v predikcii presnejší než algoritmus Vivaldi. Ale treba si uvedomiť, že výsledné chyby staníc sú závislé len na predikcii voči orientačným bodom. Tým, že systém GNP neuvažuje o väzbách medzi stanicami odpadá veľké množstvo nameraných hodnôt latencií vhodných na ďalšie optimalizovanie polohy. Z tohto dôvodu sa mi algoritmus Vivaldi javí v nasadení na daný dátový súbor pod danými simulačnými podmienkami ako presnejší.

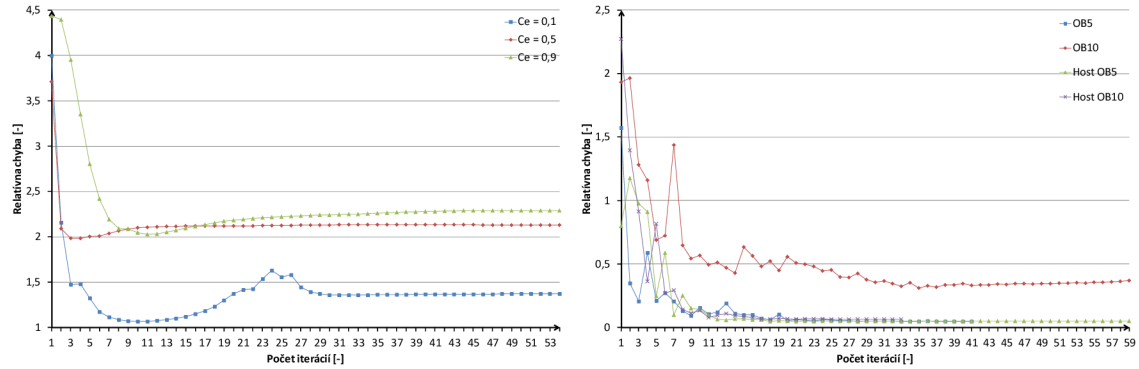
V ďalšej časti som simuloval chovanie algoritmu Vivaldi a systému GNP na podmnožinách súboru s dátami. Ako už bolo zmienené, dáta som rozdelil do geografických oblastí. Cieľom týchto testov bolo zistenie, aké výsledky budú dosahovať obidva systémy na dátach s menším počtom väzieb. Predpokladom bolo, že výsledky simulácií budú vykazovať menšiu chybu z dôvodu prítomnosti uzlov s podobnou sieťovou topológiou a tým pádom namerané latencie dvoch väzieb budú viac zodpovedať približne rovnakým vzdialenostiam. Výsledky simulácií je možné vidieť na nasledujúcich grafoch.



Obr. 4.4: Priebeh absolútnej chyby pre uzly v oblasti Ázie: a) algoritmus Vivaldi, b) systém GNP



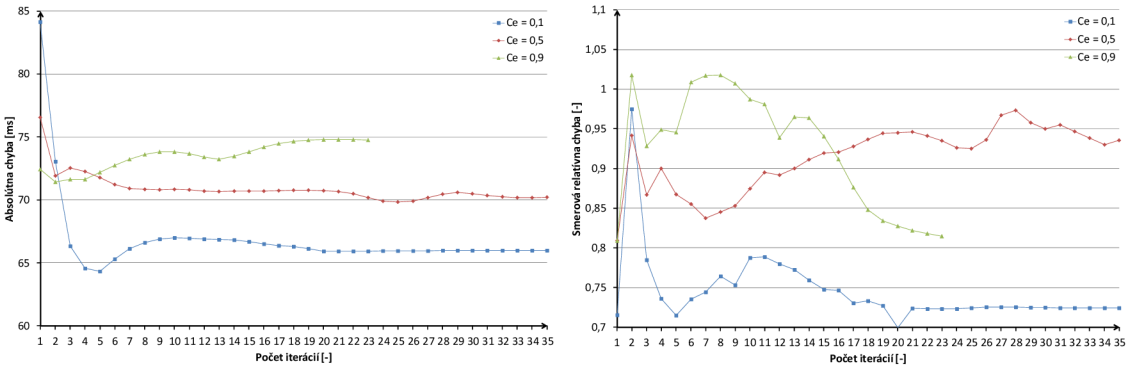
Obr. 4.5: Priebeh smerovej relatívnej chyby pre uzly v oblasti Ázie: a) algoritmus Vivaldi, b) systém GNP



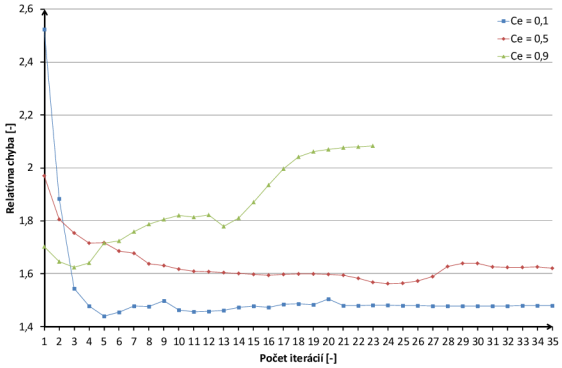
Obr. 4.6: Priebeh relatívnej chyby pre uzly v oblasti Ázie: a) algoritmus Vivaldi, b) systém GNP

Vyššie uvedené grafy zobrazujú priebeh chýb algoritmu Vivaldi a systému GNP pre uzly z oblasti Ázie. Tento dátový súbor obsahoval 20 uzlov z pôvodných 212. Hodnota absolútnej chyby pre algoritmus Vivaldi vykazuje najnižšie hodnoty pre ladiaci parameter c_e = 0,1. Takisto aj hodnota smerovej relatívnej chyby a relatívnej chyby vykazuje najnižšie hodnoty pre túto hodnotu parametru c_e. Pravdepodobným vysvetlením s ohľadom na relatívne malý počet konvergujúcich uzlov a ich vzájomne nízku vzdialenosť pri inicializácii náhodných súradníc (z dôvodu rozsahu generovaných súradníc) je vplyv veľkých výkyvov chýb uzlov e_i pri vyšších hodnotách c_e na osciláciu siete. Čo sa týka systému GNP, hodnota absolútnej chyby dosahuje 5 milisekúnd pre päť OB a 22 milisekúnd pre desať OB. Ako vysvetlením je už spomenutý fakt, že pre vyšší počet orientačných bodov dochádza ku kumulácii priemeru absolútnej chyby. Podobný priebeh majú aj grafy smerovej relatívnej chyby a relatívnej chyby. Pre stanice sa drží absolútna chyba v obidvoch prípadoch (5 alebo 10 OB) na podobnej úrovni, z čoho vyplýva fakt, že počet orientačných bodov nemá

vplyv na výslednú hodnotu. Podobný charakter majú aj priebehy smerovej relatívnej a relatívnej chyby, z čoho vyplýva rovnaký záver.

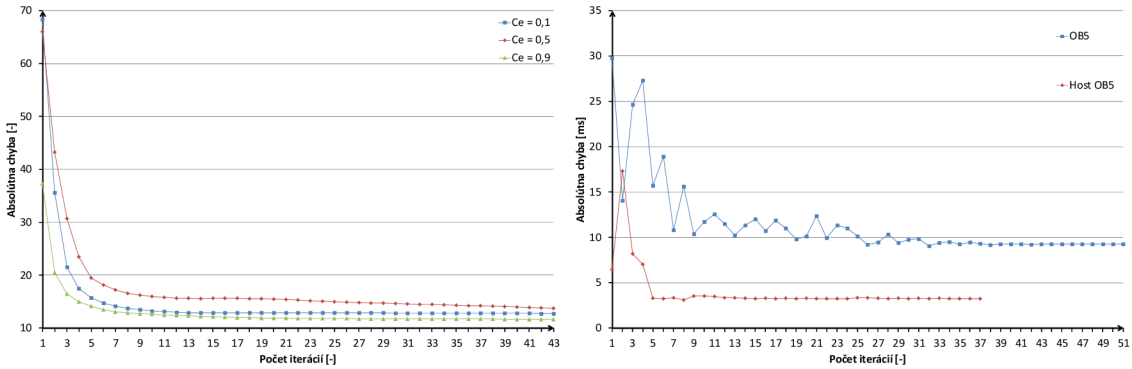


Obr. 4.7: Priebeh chýb algoritmu Vivaldi pre uzly v oblasti Severnej Ameriky: a) absolútna chyba, b) smerová relatívna chyba

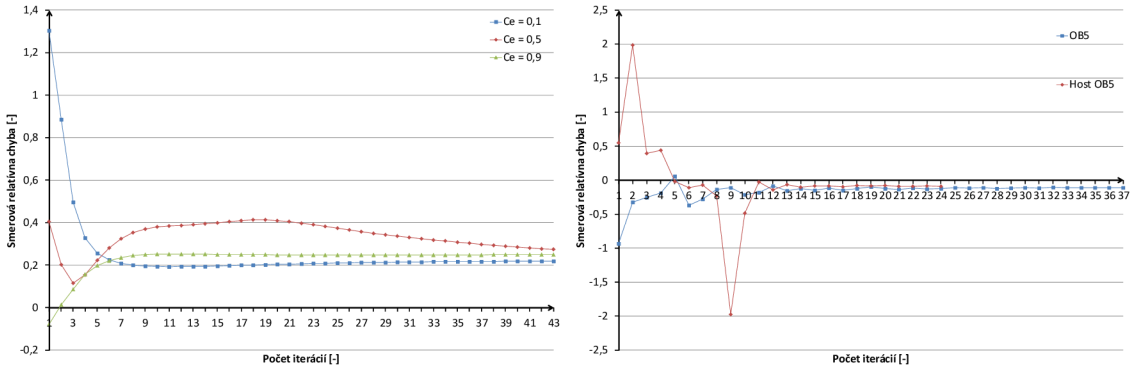


Obr. 4.8: Priebeh relatívnej chyby algoritmu Vivaldi pre uzly v oblasti Severnej Ameriky

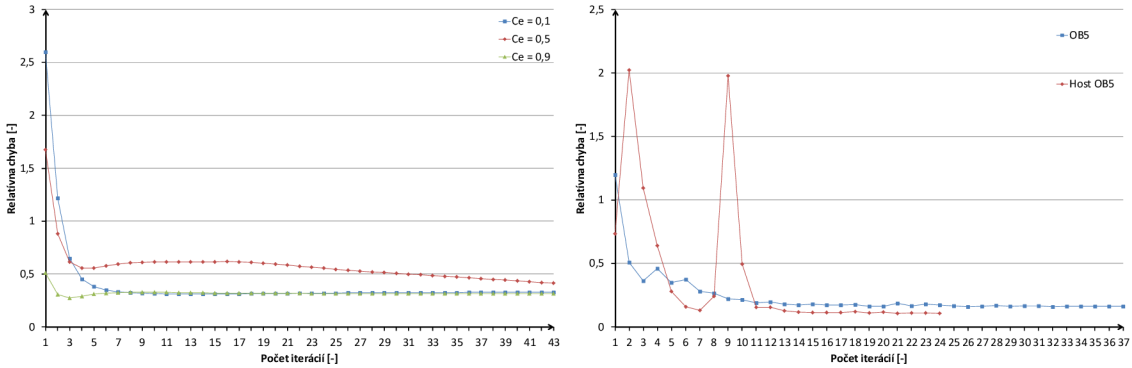
Vyššie uvedené grafy zobrazujú priebeh chýb algoritmu Vivaldi pre uzly z oblasti Severnej Ameriky. Pre systém GNP sa mi nepodarilo zmerať priebehy jednotlivých chýb. Tento dátový súbor obsahoval 105 uzlov z pôvodných 212. Počet záznamov hodnoty RTT bol 4600, z čoho vyplýva, že jeden uzol mal v priemere 43 väzieb k ďalším uzlom. Hodnota absolútnej chyby vykazuje najnižšie hodnoty pre ladiaci parameter c_e = 0,1. Takisto aj hodnota smerovej relatívnej chyby a relatívnej chyby vykazuje najnižšie hodnoty pre túto hodnotu parametru c_e.



Obr. 4.9: Priebeh absolútnej chyby pre uzly v oblasti Južnej Ameriky: a) algoritmus Vivaldi, b) systém GNP



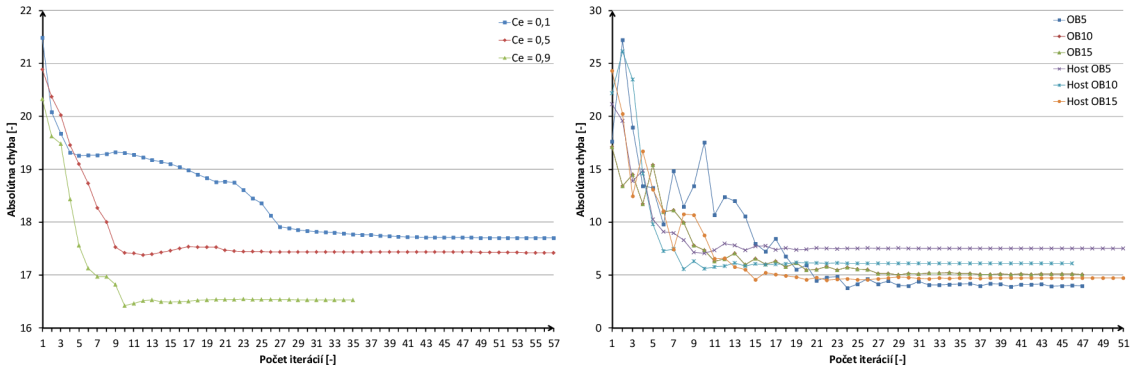
Obr. 4.10: Priebeh smerovej relatívnej chyby pre uzly v oblasti Južnej Ameriky: a) algoritmus Vivaldi, b) systém GNP



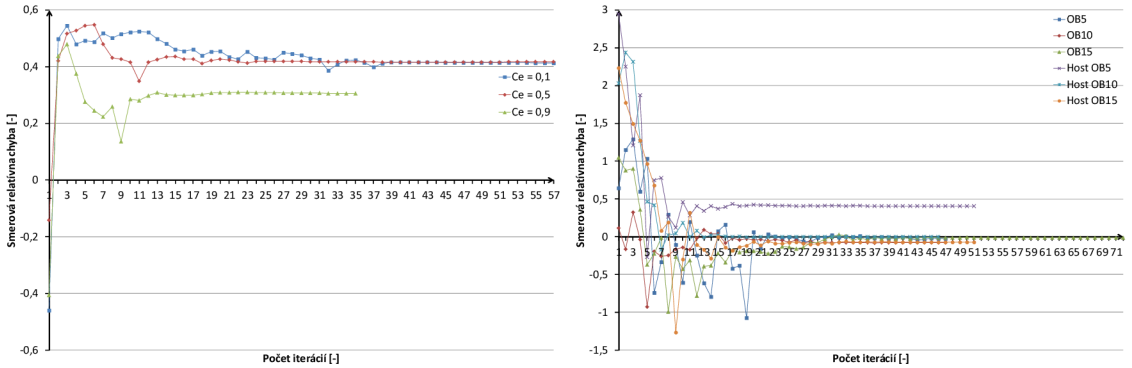
Obr. 4.11: Priebeh relatívnej chyby pre uzly v oblasti Južnej Ameriky: a) algoritmus Vivaldi, b) systém GNP

Vyššie uvedené grafy zobrazujú priebeh chýb algoritmu Vivaldi a systému GNP pre uzly z oblasti Južnej Ameriky. Tento dátový súbor obsahoval len 7 uzlov z pôvodných 212.

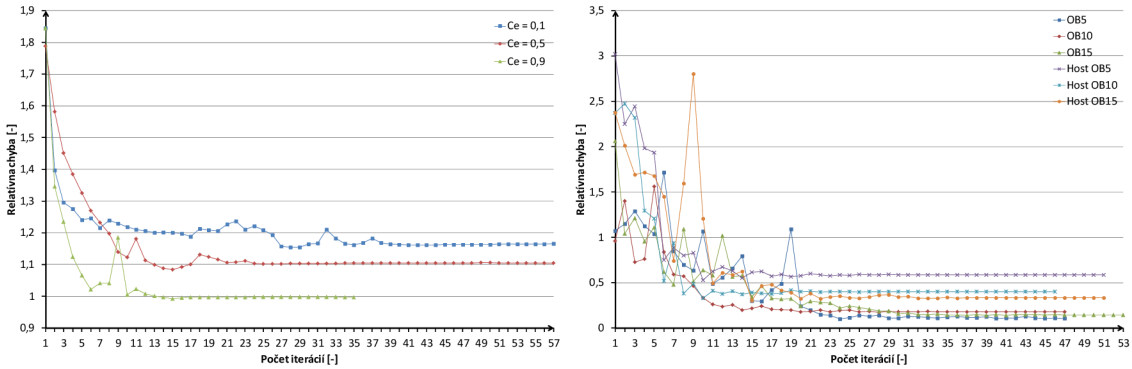
Počet záznamov hodnoty RTT bol 20, z čoho vyplýva, že jeden uzol mal v priemere 3 väzby k ďalším uzlom. Počet záznamov bol v tomto prípade veľmi nízky a tak priebehy jednotlivých chýb pri nastavení rôznych ladiacich parametrov algoritmu Vivaldi alebo rôznom počte orientačných bodov systému GNP dosahujú temer zhodné hodnoty. Tým pádom je možné tvrdiť, že pri nízkom počte uzlov (do 10) nemá nastavenie ladiacich parametrov alebo počet orientačných bodov vplyv na proces lokalizácie uzlov ani jedného z algoritmov.



Obr. 4.12: Priebeh absolútnej chyby pre uzly v oblasti Európy: a) algoritmus Vivaldi, b) systém GNP



Obr. 4.13: Priebeh smerovej relatívnej chyby pre uzly v oblasti Európy: a) algoritmus Vivaldi, b) systém GNP



Obr. 4.14: Priebeh relatívnej chyby pre uzly v oblasti Európy: a) algoritmus Vivaldi, b) systém GNP

Vyššie uvedené grafy zobrazujú priebeh chýb algoritmu Vivaldi a systému GNP pre uzly z oblasti Európy. Tento dátový súbor obsahoval 80 uzlov z pôvodných 212. Počet záznamov hodnoty RTT bol 2800, z čoho vyplýva, že jeden uzol mal v priemere 35 väzieb k ďalším uzlom. Hodnota absolútnej chyby pre algoritmus Vivaldi vykazuje najnižšie hodnoty pre ladiaci parameter c_e = 0,9. Takisto aj hodnota smerovej relatívnej chyby a relatívnej chyby vykazuje najnižšie hodnoty pre túto hodnotu parametru c_e. V tomto prípade je situácia rozdielna než v prípade s uzlami z oblasti Ázie. Vyšší počet väzieb má za následok rýchlejšiu a presnejšiu konvergenciu pri vyššej hodnote c_e. Čo sa týka systému GNP, hodnota absolútnej chyby pre orientačné body sa ustálila na piatich milisekundách. Takisto aj smerová relatívna chyba sa ustálila na hodnote -0,1 a relatívna chyba na hodnote 0,2 pre všetky tri merania. Z tohto vyplýva, že vyšší počet väzieb s podobnými parametrami ako je oneskorenie paketov, tvorba front na aktívnych sieťových prvkoch alebo jitter majú za následok presnejšiu predikciu v dvojrozmernom priestore.

5 ZÁVER

Táto diplomová práca bola zameraná na oboznámenie sa s algoritmami Vivaldi a GNP, naštudovanie ich vlastností a vytvorenie simulačnej knižnice pre prácu s RTT odozvami nameranými v sieti PlanetLab. Vivaldi je jednoduchý distribuovaný decentralizovaný algoritmus, ktorý využíva umelé súradnicové systémy k odhadu latencií medzi stanicami v sieti. Naproti tomu algoritmus GNP je jednoduchý centralizovaný algoritmus, ktorého myšlienkou je modelovať Internet ako geometrický priestor. Sieťová vzdialenosť je následne odhadovaná modelom geometrickej vzdialenosti medzi nimi.

Ďalšou časťou tejto diplomovej práce bolo vyvinúť simulačnú knižnicu pre porovnanie týchto algoritmov a vyhodnotenie ich schopnosti predikcie latencie, a ktorá je zároveň schopná pracovať s databázou RTT odoziev nameranými v sieti PlanetLab. K tomuto bol použitý programovací jazyk Java z dôvodov platformovej prenositeľnosti (t.j., použiteľnosť na ľubovoľnom operačnom systéme) ako aj dostupnosti nástrojov na tvorbu grafického užívateľského rozhrania.

Pri simuláciách som zistil, že obidva algoritmy dosahujú celkom prijateľné hodnoty chýb predikcie, avšak pri zachovaní určitých podmienok. Algoritmy sa v použitom 2-rozmernom priestore chovali podľa predpokladov, tzn. že porušenie trojuholníkov nerovnosti vnášalo značnú chybu do celého systému. Z tohto dôvodu je možné tvrdiť, že nie je možné efektívne porovnať schopnosti predikcie algoritmu Vivaldi so systémom GNP v 2-rozmernom priestore, kde jediným parametrom popisujúcim väzbu medzi dvomi uzlami je latencia. Aby sme boli schopný dobre porovnať a vyhodnotiť obidva systémy, je potrebné aby každý z nich pracoval v priestore, kde dosahuje najlepšie výsledky. To znamená priestor, ktorý modeluje Internet, by mal brať v úvahu viaceré parametre na ceste medzi dvomi uzlami (napr. oneskorenie, jitter, tvorbu front, ...) ako svoje rozmery a týmto parametrom prikladať určitú váhu na vplyv posunu uzlov v tomto priestore pri hľadaní polohy s čo najmenšou chybou.

LITERATÚRA

[1] Dabek, F., Cox, R., Kaashoek, F., Morris, R., aj.: *Vivaldi: A Decentralized Network Coordinate System.* In Proceedings of ACM SIGCOMM 2004 [online], Portland: Massachusetts Institute of Technology, 2004. URL: <http://pdos.csail.mit.edu/papers/vivaldi:sigcomm/paper.pdf>

[2] Cox, R., Dabek, F., Kaashoek, F., Li, J., Morris, R.; aj.: *Practical, Distributed Network Coordinates*, In Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II) [online], Cambridge (USA): Massachusetts Institute of Technology. URL: <http://pdos.csail.mit.edu/~rtm/papers/hotnets-vivaldi.pdf>

[3] Chen, Y., Xiong, Y., Shi, X., aj.: *Pharos: A Decentralised and Hierarchical Network Coordinate System for Internet Distance Prediction*. In Proceeding of the 50th Annual IEEE Global Telecommunications Conference [online], Washington, D.C., USA, November 2007.
URL: <http://www.net-glyph.org/~chenyang/Globecom07_Pharos.pdf>

[4] Chen, Y., Xiong, Y., Shi, X., Zhu, J., Deng, B., Li, X.; aj.: *Pharos: Accurate and Decentralized Network*. In Proceeding of the IEEE GLOBECOM 2007 [online], Washington, D.C., USA, November 2007.
URL: <http://learn.tsinghua.edu.cn:8080/2004310399/IET_Pharos.pdf>

[5] Chen, Y., Zhao, G., Li, A.,; aj.*: Myth: An Accurate and Scalable Network Coordinate System under High Node Churn Rate*. In Proceeding of the 15th IEEE International Conference on Networks [online], Adelaide, Australia, November 2007.
URL: <http://www.net-glyph.org/~chenyang/ICON07_Myth.pdf>

[6] Dabek, F., Li, J., Sit, E.; aj.: *Designing a DHT for low latency and high throughput*. In NSDI'04: Proceeding of the 1st conference on Symposium on Networked Systems Design and Implementation [online], Berkley, CA, USA: USENIX Association 2004.
URL: <http://pdos.csail.mit.edu/papers/dhash:nsdi/paper.pdf>

[7] Ng, T., Zhang, H.: *Predicting Internet network distance with coordinate-based approaches*. In INFOCOM 2002. 21st Annual Joint Conference of the IEEE Computer and Communications Societes [online]. New York, NY, USA, June 2002.
URL: <http://www.cs.rice.edu/~eugeneng/papers/INFOCOM02.pdf>

[8] Donnet, B., Gueye, B., Kaafar, M. A.; aj.: *A Survey on Network Coordinates Systems, Design and Security* [online]. CSE Department, Univ. Catholique de Louvain, Louvain-la-Neuve, Belgium, May 2010.
URL: <ftp://ftp.run.montefiore.ulg.ac.be/pub/RUN-PP09-08.pdf>

[9] Theory.org Wiki. *Bittorrent Protocol Specification v1.0* [online].
URL: <http://wiki.theory.org/BitTorrentSpecification>

[10] Li-wei, L.; aj.: *A Decentralized Network Coordinate System for Robust Internet Distance Prediction* [online]. Cambridge (USA): Massachusetts Institute of Technology, Department of Civil and Environmental Engineering, 2005.
URL: <http://web.mit.edu/lilehman/www/paper/pcoord05.pdf>

[11] Chu, Y.H., Ganjam, A., Ng, T.S.E., Rao, S.G., Sripanidkulchai, K., Zhan, J., Zhang, H.; aj.: *Early Deployment experience with an overaly based Internet broadcasting system* [online]. In Proceeding of USENIX Annual Technical Conference, Jun 2004.
URL: <http://www.cs.cmu.edu/~sanjay/Papers/techreport.pdf>

[12] Godfrey, P.B., Shenker, S., Stoica, I.; aj.: *Minimizing Churn in Distributed Systems* [online]. In Proceeding of ACM SIGCOMM, September 2006.
URL: <http://www.cs.berkeley.edu/~istoica/papers/2006/churn.pdf>

[13] Ng, T.S.E., Zhang, H.; aj.: A Network Positioning System for the Internet [online]. In Proceeding of USENIX Annual Technical Conference, Boston, MA, Jun 2004.
URL: <http://www.cs.rice.edu/~eugeneng/papers/USENIX04.pdf>

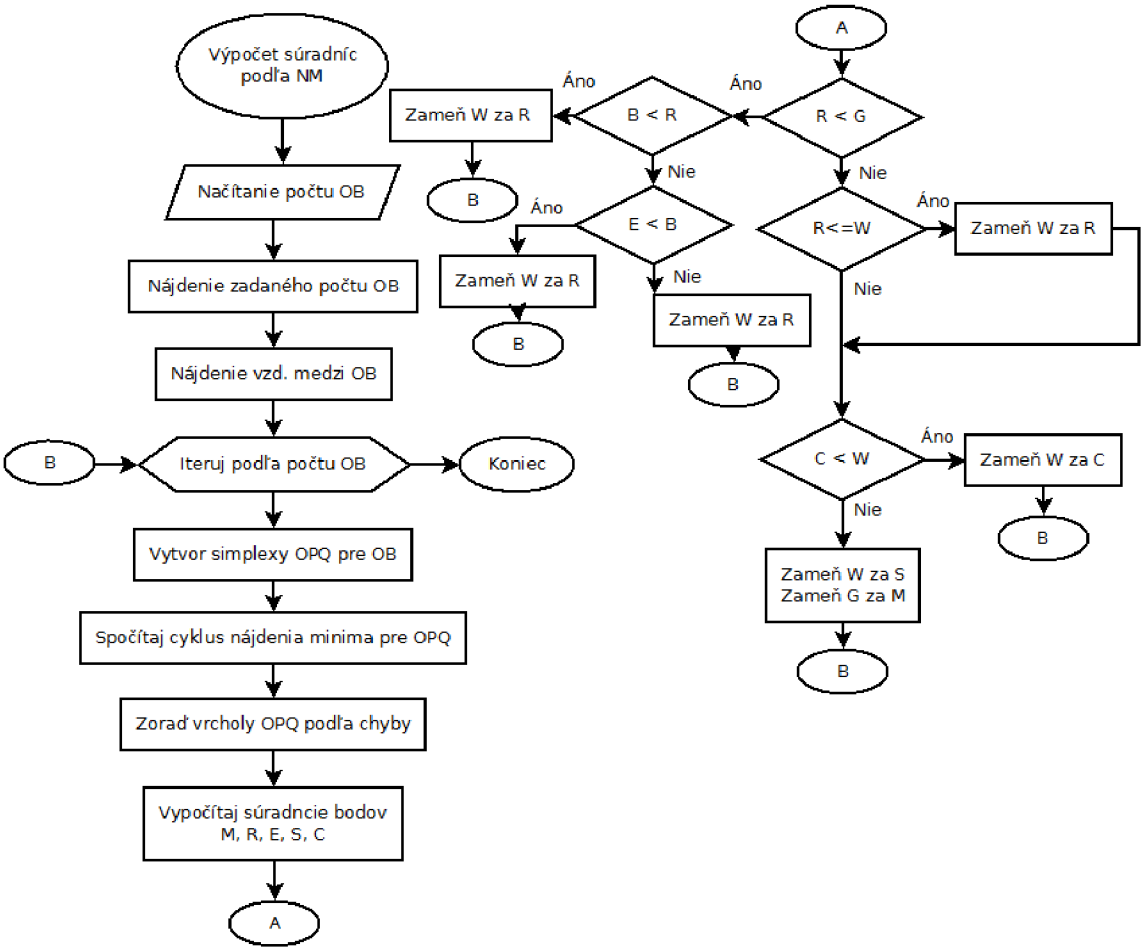
ZOZNAM SYMBOLOV, VELIČÍN A SKRATIEK

RTT	čas vybavenia dotazu – Round-Trip Time
GNP	globálne sieťové polohovanie – Global Network Positioning
P2P	rovný s rovným – Peer-to-Peer
ICMP	Internet Control Message Protocol – základný signalizačný protokol sady TCP/IP
OB	orientačný bod
DNS	Domain Name System
DRE	smerová relatívna odchýlka – Directional Relative Error

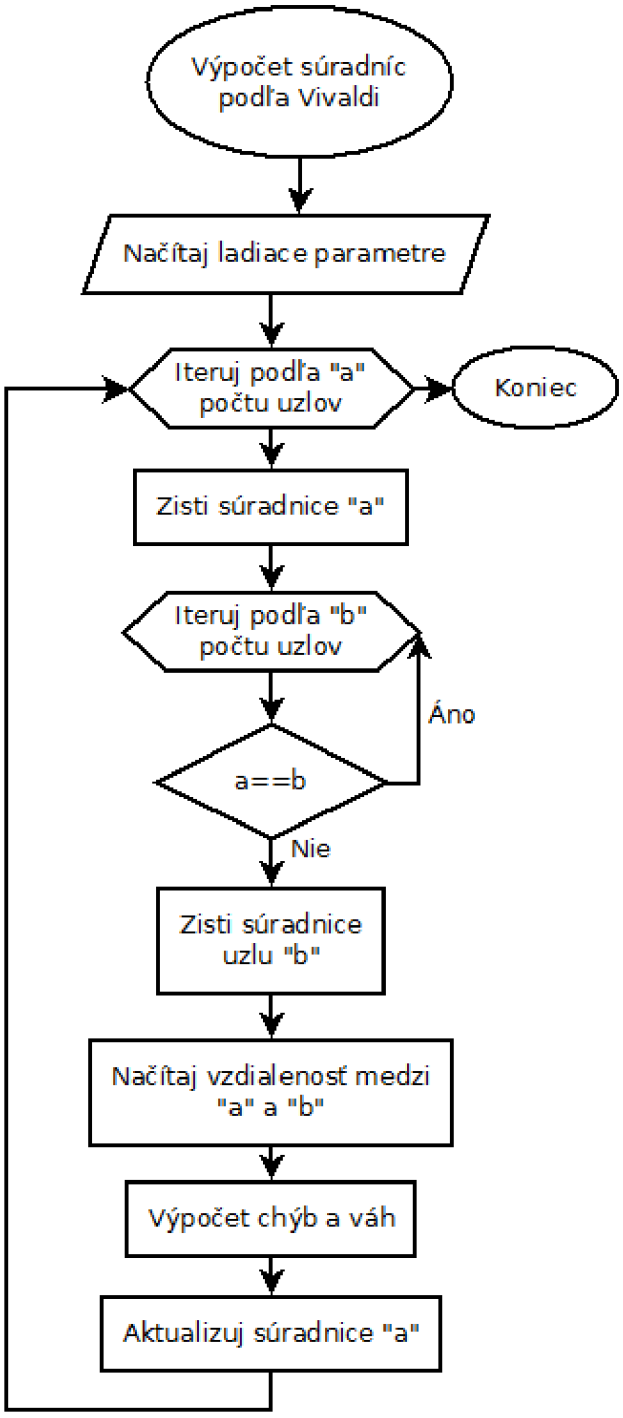
ZOZNAM PRÍLOH

A VÝVOJOVÉ DIAGRAMY

A.1 Vývojový diagram metódy Nelder-Mead



A.2 Vývojový diagram algoritmu Vivaldi



B OBSAH CD

Zložka VivaldiGNP-lib	*obsahuje zdrojové kódy vyvinutej knižnice*
Súbor xsulik00-diplomka.docx	*diplomová práca vo formáte *.docx*
Súbor xsulik00-diplomka.pdf	*diplomová práca vo formáte *.pdf*