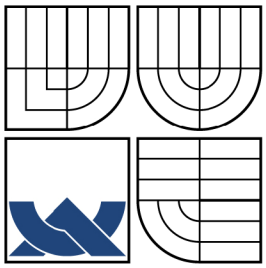
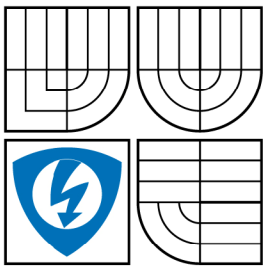


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

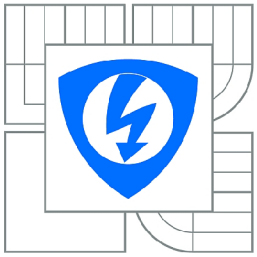
LOKALIZACE STANIC V INTERNETU POMOCÍ METODY VIVALDI S ADAPTIVNÍM ČASOVÝM KROKEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. JAN MAŠÍN
AUTHOR

VEDOUCÍ PRÁCE doc. Ing. DAN KOMOSNÝ, Ph.D.
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. Jan Mašín ***ID:*** 77842
Ročník: 2 ***Akademický rok:*** 2010/2011

NÁZEV TÉMATU:

Lokalizace stanic v Internetu pomocí metody Vivaldi s adaptivním časovým krokem

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s principy vyhodnocování logické polohy stanic v síti Internet. Prostudujte lokalizační algoritmus s názvem Vivaldi a následně proveďte jeho realizaci v operačním systému GNU/Linux distribuce CentOS. Seznamte se s experimentální sítí PlanetLab http://www.planet-lab.org/. Na zvolené stanice této sítě přeneste vytvořenou aplikaci a ověřte její činnost na reálných serverech rozmístěných na různých místech zeměkoule. Zhodnoťte dosahovanou přesnost a rychlost odhadu vzdáleností mezi stanicemi v síti PlanetLab.

DOPORUČENÁ LITERATURA:

[1] DABEK, F., COX, R., KAASHOEK, F., MORRIS, R. Vivaldi: a decentralized network coordinate system. ACM SIGCOMM Computer Communication Review. Association for Computing Machinery, 2004. ISSN: 0146-4833.
[2] COX, R. DABEK, F. Learning Euclidean coordinates for Internet hosts. ACM SIGCOMM Computer Communication Review. Association for computing machinery, 2004. ISSN: 0146-4833.
[3] PlanetLab Consortium. PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services [online]. URL: <http://www.planet-lab.org> [cit. 13. 10. 2009].

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

Vedoucí práce: doc. Ing. Dan Komosný, Ph.D.

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

UPOZORNĚNÍ:

ANOTACE

Cílem této diplomové práce bylo seznámení se s principy vyhodnocování logické polohy stanic v síti Internet. Prostudovat lokalizační algoritmus s názvem Vivaldi s adaptivním časovým krokem a následně provést jeho realizaci v operačním systému GNU/Linux distribuce CentOS. Seznámit se s experimentální sítí PlanetLab (*http://www.planet-lab.org/*). Na zvolené stanice z této sítě přenést vytvořenou aplikaci a ověřit její činnost na reálných serverech rozmístěných na různých místech zeměkoule a zhodnotit dosahované přesnosti odhadu vzdáleností mezi stanicemi v síti PlanetLab.

V rámci této práce byla vytvořena aplikace pro měření predikce zpoždění za pomoci algoritmu Vivaldi s adaptivním časovým krokem, která funguje na principu klient-server, kde klientská část provádí kroky algoritmu vivaldi a serverová pouze naslouchá, sbírá výsledná data algoritmu Vivaldi a přehledně je ukládá do souboru. Dále pak byla vytvořena i aplikace pro přímé měření zpoždění, která funguje také jako model klient-server. Tyto aplikace byly přeneseny na vybrané uzly z experimentální sítě PlanetLab. Následně pak byly na těchto uzlech spuštěny pro uskutečnění potřebného měření.

Výsledné hodnoty byly zpracovány do tabulek za pomoci programu Microsoft Excel. Tyto hodnoty pak byly porovnány s přímým měřením a s konkurenční lokalizační metodou King. Lokalizační metody Vivaldi s adaptivním časovým krokem a King, se porovnávaly na základě vypočtených relativních chyb obou odhadů měření a pomocí distribučních funkcí všech relativních chyb obou metod. Všechny tyto informace byly vyhodnoceny k porovnání přesností obou lokalizačních metod a přímého měření.

Klíčová slova: lokalizační metody, algoritmus Vivaldi, metoda King, CentOS, PlanetLab

ABSTRACT

The aim of this thesis was to identification with the principles of logical evaluation of the position of stations on the Internet. Read up on the localization algorithm called Vivaldi with adaptive time step and subsequently to its implementation in the operating system GNU/Linux CentOS distribution. Do one's homework the PlanetLab experimental network (*http://www.planet-lab.org/*). At selected stations from the network transfer created by the application and verify its function on the real servers located at various places around the globe and assess the accuracy achieved by estimating the distance between stations on the PlanetLab network.

In this scope of activity, the application was created to measure the delay prediction using Vivaldi algorithm with adaptive time step which is on principle of operation a client-server where the client performs the steps of the algorithm, Vivaldi and the server only listens, collects the resulting data Vivaldi algorithm and stores them neatly file. Furthermore, the application was developed for direct measurement of the delay, which also functions as a client-server. These applications have been transferred to the selected nodes from the PlanetLab experimental network. Subsequently, these nodes were running, to carry out the necessary measurements.

The resulting values were work into tables of using Microsoft Excel. These values were then compared with direct measurements and competitive positioning by the King. Vivaldi localization methods with adaptive time step and the King, were compared based on calculated estimates of both real estate errors and measurement using distribution function of the relative errors of both methods. All this information was evaluated to compare accuracy of both the localization methods and direct measurements.

Crucial words: methods location, algorithm Vivaldi, method King, CentOS, PlanetLab

MAŠÍN, J. Lokalizace stanic v Internetu pomocí metody Vivaldi s adaptivním časovým krokem. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2011. 57 s. Vedoucí diplomové práce doc. Ing. Dan Komosný, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svojí diplomovou práci na téma „Lokalizace stanic v internetu pomocí metody Vivaldi s adaptivní časovým krokem“ jsem vypracoval samostatně pod vedením vedoucího semestrální 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 semestrální 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 19.5.2011 ……..

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce doc. Ing. Danu Komosnému, Ph.D. za vedení, pomoc a užitečné rady při zpracovávání diplomové práce.

V Brně dne 19.5.2011 .....

Seznam použitých zkratek

CentOS	Community ENTerprise Operating System
CESNET	Czech Education and Research Network
GNP	Global Networ Positioning
ICMP	Internet Control Message Protocol
IDMaps	Internet Distance Map service
IP	Internet Protokol
IPv4	Internet Protokol verze 4
IPv6	Internet Protokol verze 6
IPTV	Internet Protocol Television
LSB	Linux Standard Base
PING	Packet InterNet Groper
PlMan	PlanetLab experiment manager
RPM	RPM Package Manager
RTT	Round-Trip Time
SSH	Secure SHell
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol

Obsah

Seznam obrázků

Seznam tabulek

1 ÚVOD

V posledních letech dochází k prudkému navýšení počtu stanic v celosvětové síti Internet a s tím jsou spojeny i jisté komplikace, jako například nedostatečný rozsah IP (Internet Protokol) adres v IP verze 4 (IPv4), který byl částečně vyřešen (oddálen) dělením sítí na podsítě, využíváním virtuálních sítí a zavedením nového protokolu Internet protokol verze 6 (IPv6). S narůstajícím počtem stanic v Internetu rostou i nároky na efektivitu datové komunikace, služby v reálném čase a tím nabývá na důležitosti i problematika lokalizace stanic v Internetu.

Tato diplomová práce se zabývá právě touto problematikou. Pro řadu aplikací není ani tak důležité fyzické umístění stanice, jako spíš kvalita přenosové cesty. Nejjednodušším způsobem lokalizace je zjištění odezvy mezi stanicemi např. nástrojem *ping* (Packet InterNet Groper). Tímto postupem se však musí zjišťovat odezva mezi všemi zúčastněnými uzly a to je velmi neefektivní způsob lokalizace.

Proto existují metody lokalizace snažící se o využití co nejnižších výpočetních a síťových nároků pro nalezení pozice jednotlivých stanic v Internetu. Ať už to jsou metody využívající umělých souřadnicových systémů, jako je např. metoda Vivaldi, která je řešena v této diplomové práci, tak metody využívající přímých měření jako např. metoda King či systém Meridian. Podrobnější popis problematiky lokalizace stanic v Internetu a jednotlivých metod lokalizace bude uveden dále v textu.

Cílem diplomové práce je se seznámit s principy vyhodnocování logické polohy stanic v celosvětové síti Internet, kde logická poloha je vyjádřena souřadnicemi v určitém umělém souřadnicovém prostoru, které vyjadřují vzdálenost (odezvu) od ostatních stanic v dané síti, oproti tomu fyzická poloha vyjadřuje geografické (fyzické) umístění uzlů na zemi. Prostudovat lokalizační algoritmus Vivaldi s adaptivním časovým krokem, promyslet jeho realizaci v operačním systému GNU/Linux distribuce CentOS a seznámit se s experimentální síti PlanetLab, která je přístupná z *http://www.planet-lab.org/*. Dále pak realizovat a implementovat algoritmus Vivaldi s adaptivním časovým krokem do experimentální sítě PlanetLab. Na zvolené stanice v této experimentální síti přenést vytvořenou aplikaci a ověřit její činnost na reálných serverech rozmístěných na různých místech planety. Dále pak zhodnotit přesnost odhadu vzdáleností mezi stanicemi v síti PlanetLab a porovnat tyto parametry s vybranou lokalizační metodu King. Tuto metodu realizuje v rámci diplomové práce kolega Bc. Michal Exler (Lokalizace stanic v internetu pomocí metody King).

2 TEORETICKÝ ROZBOR

V této kapitole je uveden teoretický rozbor týkající se problematiky lokalizace stanic v internetu.

2.1 Význam určování polohy

Přenos dat k jednotlivým uzlům s využitím používaných protokolových sad, je v dnešní době poměrně jednoduché. Logické spojení se vzdálenou stanicí je zprostředkováno pomocí aplikace běžící na jednom uzlu, která využívá rozhraní dané protokolové sady. Nejrozšířenější protokol je dnes TCP/IP (Transmission Control Protocol - Internet Protocol), který pracuje na úrovni transportní vrstvy [2]. Na fyzické vrstvě protokolu TCP/IP odpovídá logickému spoji více přenosů po fyzických datových linkách. Pro řadu aplikací je velmi výhodné využití logických tras, pomocí kterých zajišťují komunikaci v síti. Jde hlavně o služby, které jsou založené na překryvných sítích [18].

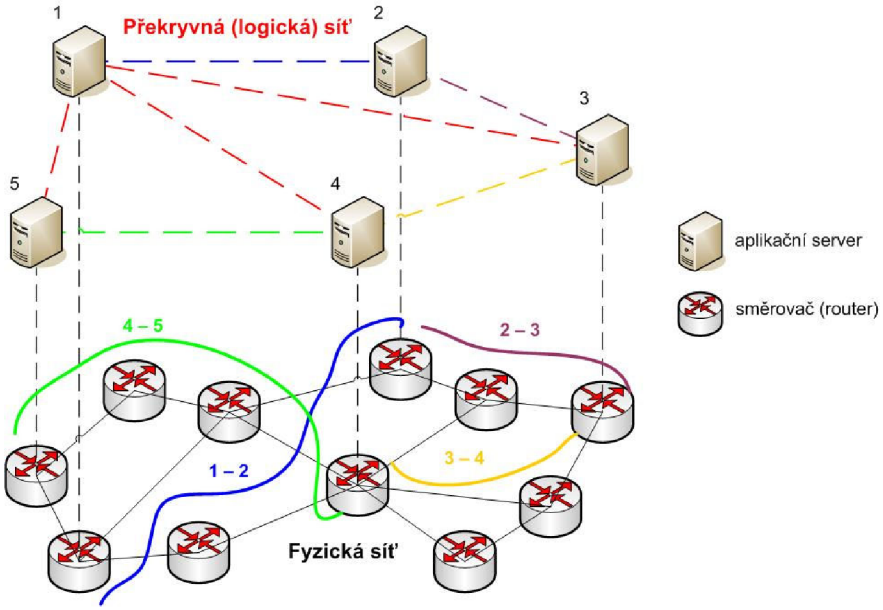
Význam určování polohy stanic v Internetu spočívá v nalezení nejlepších možných cest pro datovou komunikaci mezi jednotlivými uzly. Pro většinu aplikací je důležité, aby znaly šířku pásma, stálost (nestálost) zpoždění či odezvu v přenosovém datovém kanálu. Pro datovou komunikaci je vhodné využít trasy s co nejširším přenosovým pásmem. Samotná znalost fyzické topologie sítě však nezaručuje, že přenos mezi danými stanicemi, který je z geografického hlediska nejvýhodnější, bude právě ten nejefektivnější. Stanice, které si jsou fyzicky nejblíže, mohou mít slabé přenosové kanály, a naopak u vzdálenějších stanic může být šířka pásma přenosového kanálu mnohem větší. Z těchto důvodů je často výhodnější určit polohu stanic v Internetu pro jejich následnou komunikaci podle těchto kritérií.

2.2 Absolutní a relativní poloha

Pojem poloha počítače v internetu se dá chápat dvěma způsoby, jednak jako fyzická pozice stanice (geografická poloha) v rámci celého světa, tzv. absolutní pozice. Druhou možností je relativní poloha, která vyznačuje pouze logickou pozici počítače v síti. Jak již bylo řečeno v úvodu, z důvodu prudkého navýšení počtu stanic v Internetu v posledních letech nabývá na důležitosti právě relativní určení polohy pro aplikace využívajících překryvných sítí. Zmíněné aplikace, na základě znalosti relativní pozice dané stanice,

mohou pracovat efektivněji, a tím dochází i k ušetření značných síťových prostředků. To může být výhodné při výběru vhodných zdrojů a příjemců ve stromové struktuře při hromadné komunikaci u IPTV (Internet Protocol Television) [1].

U stanic překryvných sítí probíhá komunikace po logických linkách. Na obrázku Obr 2.1 jsou znázorněny logické cesty, které se mohou výrazně lišit od fyzického propojení jednotlivých stanic. Aby aplikace využívající překryvných sítí mohly efektivně fungovat, je třeba, aby obsahovaly metody pro určení relativní pozice stanice v Internetu. Aplikace tedy musí obsahovat informace o vlastnostech komunikační sítě, jako je např. velikost zpoždění mezi uzly nebo šířku pásma přenosového kanálu [19].



Obr 2.1: *Příklad překryvné (logické) sítě a její fyzické topologie.*

2.3 Zpoždění v síti

Zpoždění mezi dvěma stanicemi se dá chápat jako doba od odeslání paketu zdrojovou stanicí a jeho přijetím v cílové stanici. V tom případě by se však jednalo o zpoždění jednosměrné, což není v praxi příliš použitelné. Obecně je vyžadována dobrá časová synchronizace obou stanic. Z těchto důvodů se více používá zpoždění obousměrné. Toto měření vyjadřuje dobu přenosu paketu od zdrojové stanice k cílové a zpět (tato doba samozřejmě zahrnuje i dobu zpracování paketu jednotlivými síťovými prvky) a je reprezentováno parametrem RTT (Round-Trip Time).

K jakékoliv stanici je možné změřit RTT pomocí signalizačních zpráv protokolu ICMP (Internet Control Message Protocol), který je součástí protokolové sady TCP/IP. Obousměrné zpoždění měří např. jeden ze základních nástrojů správy sítě ping [3]. Výsledky měření je však třeba dále statisticky zpracovat, protože ICMP paket může být směrován zpět ke zdrojové stanici různými cestami. Mezi všemi komunikujícími stanicemi není možné hodnotu RTT změřit (např. sítě peer to peer). Počet měření by totiž rostl s druhou mocninou jejich počtu. Pak by se muselo provést *N(N-1)* měření pro N koncových stanic [1].

2.4 Metody predikce zpoždění využívající přímého měření

Tyto metody nevyužívají žádných umělých souřadnicových systémů. Společným znakem všech těchto metod je snaha o odhadnutí zpoždění mezi dvěma stanicemi pomocí přímého měření mezi nimi. U takovéto predikce zpoždění s počtem stanic i kvadraticky narůstá počet měření, proto každá metoda zavádí svůj vlastní způsob, jak redukovat toto neúnosné navyšování zmíněných měření. Aby k tomuto nedocházelo, jednotlivé metody si vyberou speciální množinu stanic a s jejich pomocí určují jednotlivé vzdálenosti mezi koncovými stanicemi.

V této kapitole dále budou popsány tyto metody, využívající přímého měření pro predikci zpoždění :

- systém IDMaps (Internet Distance Map service),
- systém Internet Iso-bar,
- metoda King,
- systém Meridian.

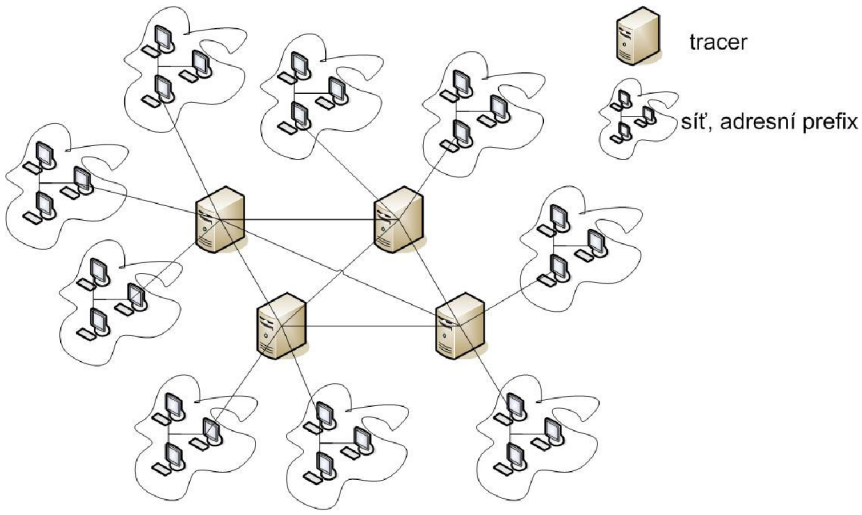
2.4.1 Systém IDMaps

IDMaps je systém pro predikci zpoždění mezi stanicemi v síti Internet, který pracuje na základě měření vzdáleností mezi stanicemi s daným adresným prefixem a jejich nejbližším tracerem. Tento sytém byl jedním z prvních systémů, který se snažil řešit již známé problémy spojené s aktivním měřením vzdálenosti v rámci všech stanic v síti [16].

Za pomoci nějakého síťové nástroje, jako např. ping či traceroute, si může koncová stanice zjistit velikost zpoždění mezi danými stanicemi. Jestliže se provádí měření zpoždění mezi velmi početnou skupinou stanic, pak dochází ke generování nadměrného množství řídících dat a dochází pak k nadbytečnému zahlcování přenosového pásma sítě. Systém IDMaps vytváří tzv. tracery a tím řeší problém s predikcí zpoždění mezi stanicemi v síti Internet. To jsou vlastně orientační uzly, které jsou vybrány jako zástupci všech stanic v síti. Hodnota zpoždění mezi koncovými stanicemi je pak určena součtem zpoždění mezi stanicí, z které měření probíhá a jejím nejméně vzdáleným tracerem, dále pak zpožděním mezi stanicí, ke které je měření určeno a jejím nejbližším tracerem a na závěr zpožděním mezi oběma tracery [19].

Systém IDMaps rozděluje Internet na menší části. Tím se razantně sníží počet měření zpoždění. Jednou z možností jak rozdělit síť na menší části je umístění orientačních bodů ke stanicím náležícím do stejného autonomního systému. Počet přiřazených tracerů je pak poměrně malý, ale zpoždění mezi stanicemi kolísá z důvodu rozsáhlosti tohoto systému. Další možností pak může být umístění orientačních bodů ke stanicím patřícím do stejného adresního prefixu. V tomto případě je počet umístěných tracerů podstatně vyšší. Výhodou však je, že stanice stejného adresního prefixu mají velmi podobné zpoždění [19].

Tento systém je tedy tvořen topologií navzájem propojených tracerů, které jsou spojeny s více autonomními systémy či adresními prefixy (viz Obr 2.2). Tracery je třeba správně umístit tak, aby měřily zpoždění s požadovanou přesností, která závisí na topologii sítě. Pro určení optimální pozice tracerů se využívají optimalizační matematické operace [19].



Obr 2.2: *Obrázek znázorňuje jednoduchý příklad sítě IDMaps.*

Orientační body (tracery), ke kterým stanice měří své zpoždění, musí být umístěny mimo jejich adresní prefix. To často vyžaduje spolupráci poskytovatelů různých služeb, což je velkou nevýhodou tohoto systému a částečně je tím zabráněno rozšiřování systému IDMaps [19].

2.4.2 Systém Internet Iso-bar

Internet Iso-bar shlukuje koncové stanice na základě vzdálenosti k daným orientačním bodům. Tyto body se však nepodílejí na odhadu vzdálenosti. Vzdálenost mezi dvěma stanicemi se určuje pomocí proměnné korelační metriky. Po nalezení vzdáleností mezi stanicemi jsou tyto stanice rozděleny do shluků (clusters), přičemž každému je určen jeho centrální bod, který pak slouží jako monitor pro daný shluk (viz Obr 2.3). Monitory jednotlivých shluků měří vzdálenosti mezi sebou, a také mezi stanicemi patřícími do shluku. Za pomoci těchto údajů pak vypočtou vzdálenost pro jakýkoliv pár stanic [17].

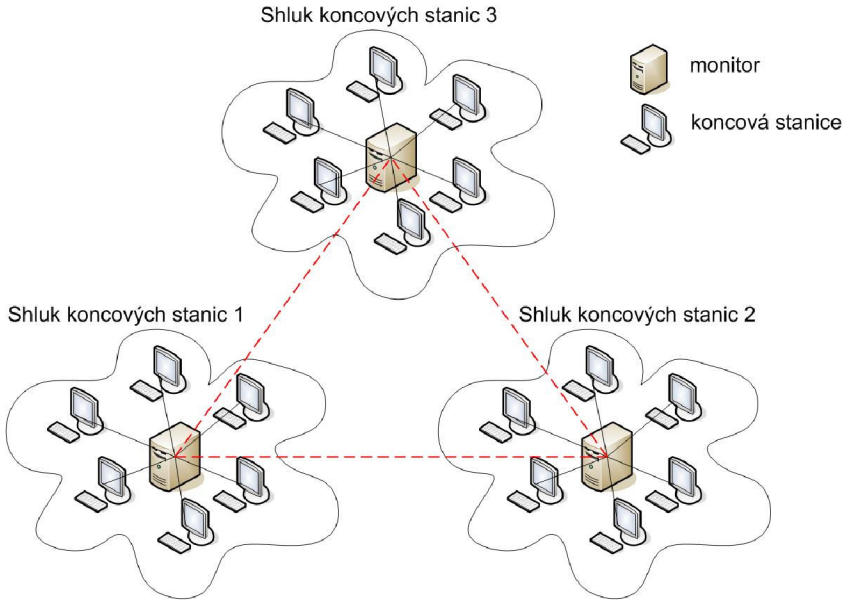
Jestliže stanice *i* a *j* patří do stejného shluku s monitorem *m*, pak pro výpočet jejich vzdálenosti platí závislost [17]:

$$L_{i,j} = \frac{L_{i,m} + L_{j,m}}{2} . \tag{2.1}$$

Pakliže stanice *i* a *j* leží v různých shlucích a je definováno, že monitor stanice *i* je m_i a stanice *j* je m_j, pak platí tento vztah [17]:

$$L_{i,j} = L_{mi,mj} . \tag{2.2}$$

Systém Internet Iso-bar poskytuje velkou míru škálovatelnosti. V peer to peer sítích se počet uživatelů může pohybovat v řádech milionů, a to znemožňuje použití centrálního bodu pro měření vzdálenosti k jednotlivým stanicím. Tento problém je řešen zmíněnou škálovatelností systému Internet Iso-bar. Dále Internet Iso-bar zjišťuje vzdálenosti téměř v reálném čase, protože nároky na výpočty jsou nízké, a také vyžaduje minimum síťové komunikace. V porovnání např. s metodou GNP dosahuje méně přesné predikce zpoždění, ale s nižší řídící komunikací v síti [17].



Obr 2.3: *Obrázek znázorňuje jednoduchý příklad sítě systému Internet Iso-bar.*

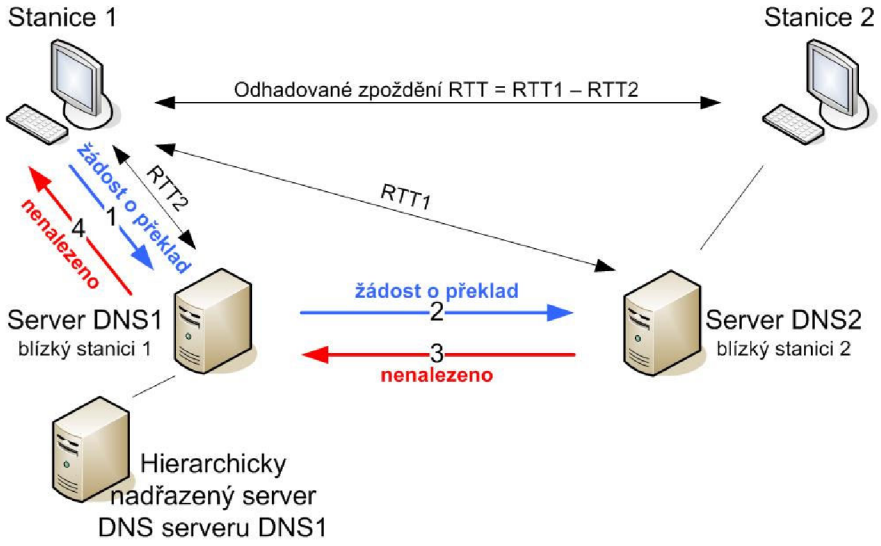
2.4.3 Metoda King

Většina stanic v internetu se nachází blízko svého DNS serveru. Zpoždění mezi stanicemi a DNS serverem je pak nízké. Mezi dvěma DNS servery je možno využít rekurzivních dotazů, které drtivá většina těchto serverů podporuje. Metoda King určuje zpoždění mezi koncovými stanicemi tak, že nalezne navzájem blízké DNS servery, které patří ke koncovým stanicím a pomocí výše zmíněných dotazů vypočítá spojení mezi DNS servery [19].

Na obrázku Obr 2.4 je znázorněno jak metoda King využívá rekurzivních dotazů k měření zpoždění mezi serverem DNS1 a DNS2. Nejdříve požádá sever DNS1 o překlad náhodného doménového jména. Doménové jméno musí patřit do domény patřící serveru DNS2. Ve vyrovnávací paměti serveru DNS1 nemusí být první pokus o překlad uložen. DNS1 bude muset poslat žádost o překlad serveru, který je hierarchicky nadřazen. Měření se tudíž musí zopakovat. Když DNS1 má ve vyrovnávací paměti záznam zapsán, předá rekurzivní dotaz serveru DNS2 a přijatou odpověď přepošle stanici 1 nazpět. Čas, který uběhne od odeslání žádosti o překlad po doručení potvrzení je označen jako RTT1. Metoda king měří zpoždění stanicí 1 a DNS1 např. pomocí pingu. Toto zpoždění je pak označeno jako RTT2. King tyto dvě zpoždění od sebe odečte a tím získá odhad zpoždění mezi servery DNS1 a DNS2 [19].

Měření je třeba provést vícekrát. Tímto opakovaným měřením pak získáme přesnější odhad zpoždění. Aby DNS1 při každém pokusu dotazoval vždy právě DNS2, je pokaždé

vygenerováno náhodně velké číslo, které reprezentuje náhodné doménové jméno v doméně serveru DNS2. Dotaz na překlad takovéhoto doménového jména v doméně serveru DNS2 bude vždy odmítnut [19].



Obr 2.4: *Princip metody King s využitím rekurzivního dotazu DNS systému.*

2.4.4 Systém Meridian

Struktura tohoto systému netvoří žádnou konkrétní strukturu a je vhodná pro moderní překryvné sítě, ale i pro stanice mimo ně. Tento systém umožňuje i složené dotazy s více podmínkami. Tak je možno zjistit stanice, které jsou v dané vzdálenosti od jiných stanic zároveň. Pro takovéto dohledávání se používá série přímých měření zpoždění. Počet těchto měření je však dostatečně malý [18].

V systému Meridian udržuje kolem sebe každá stanice určitý počet sousedních stanic. Tyto stanice jsou pak umisťovány v soustředných (nepřekrývajících se) kružnicích, jejichž poloměr exponenciálně roste. Ve vzniklých mezikružích je maximální počet stanic předem určen. Každá stanice si ukládá informace o vzdálenostech svých nejbližších sousedů, ale i informace o dostatečně velkém počtu stanic z vnějších mezikruží, které pak může v případě potřeby kontaktovat. Systém Meridian využívá pro predikci zpoždění mezi dvěma stanicemi rekurzivních dotazů, které se dotazují na vzdálenost od zdrojové stanice k cílové. Z takto získaných informací zdrojová stanice určí do jakého mezikruží patří. Dotaz se šíří ke všem stanicím v daném mezikruží. Každá dotazovaná stanice zjistí hodnotu zpoždění k cílové stanici a tuto hodnotu vrátí jako odpověď. Zdrojová stanice určí nejmenší hodnotu zpoždění z těchto odpovědí a pošle jí zpět stanici cílové [19].

Protože tento systém klade minimální nároky na topologii případné překryvné sítě, vykazuje malou režii, má nízké výpočetní nároky a velice malou chybu v porovnání se systémy, které využívají umělé souřadnicové systémy, a tak ho lze označit za perspektivní [18].

2.5 Algoritmy využívající souřadnicové systémy

Tyto algoritmy pracují na principu mapování reálné síťové topologie do umělého souřadnicového systému. Některé z nich využívají k určení souřadnic fixní množinu orientačních serverů. Jiné algoritmy, každý server připojený do sítě používají jako další orientační server s možností libovolného výběru ze všech serverů [20].

Princip těchto algoritmů tedy spočívá v přiřazení určitých souřadnic podle daného algoritmu. Vzdálenost mezi souřadnicemi dvou stanic poté reprezentuje odhadnuté zpoždění mezi nimi. Algoritmy se pak od sebe liší různými výpočty pro určení souřadnic a dále také různými druhy souřadnicových prostorů [19].

V této kapitole dále budou popsány tyto algoritmy využívající souřadnicové systémy:

- GNP - Global network positioning,
- algoritmus Lighthouses,
- systém PCoord,
- algoritmus Vivaldi.

2.5.1 Systém Global Network Positioning (GNP)

Systém GNP se snaží reprezentovat složitou strukturu Internetu v jednoduchém geometrickém prostoru, nejčastěji v N-rozměrném Euklidovském prostoru [6]. Každý host v tomto geometrickém prostoru je v síti Internet reprezentován jeho vlastními souřadnicemi, které představují jeho pozici vzhledem ke skupině orientačních bodů, tzv. Landmarks $O = O_1, O_2, O_3 .... O_N$ [7]. Tento systém vytváří matici vzdáleností L pomocí orientačních bodů, které měří hodnotu RTT mezi sebou navzájem. Jednotlivé prvky L_{ij} této matice reprezentují naměřenou vzdálenost mezi orientačními body O_i a O_j, kde $i, j = 1,2,3,...N$.

Každému orientačnímu bodu O_i je po vytvoření matice přidělen bod x_j v Euklidovském prostoru pomocí funkce minimalizace chyby [19]. Funkce je vyjádřena takto:

$$E = \sum_i \sum_j \varepsilon(L_{ij}, \|x_i x_j\|)^2 , \tag{2.3}$$

kde ε je chybová funkce. Pro optimalizační úlohu s cílem nalezení extrému funkce využívá systém GNP různých algoritmů, jako je např. Simplex Downhill [8]. Tento algoritmus pracuje tak, že pro dvě proměnné (např. souřadnice x a y) je tvořen trojúhelníkem a algoritmus pak porovnává hodnoty funkcí na jeho vrcholech. Vrchol, ve kterém má trojúhelník nejvyšší hodnotu funkce pro určité souřadnice x a y se označuje jako nejhorší vrchol. Tento vrchol je pak přesunu tak, aby se dosáhlo nižší hodnoty funkce v bodě a je vytvořen nový trojúhelník. Proces se opakuje dokud není nalezeno minimum zadané funkce s určitou tolerancí [9].

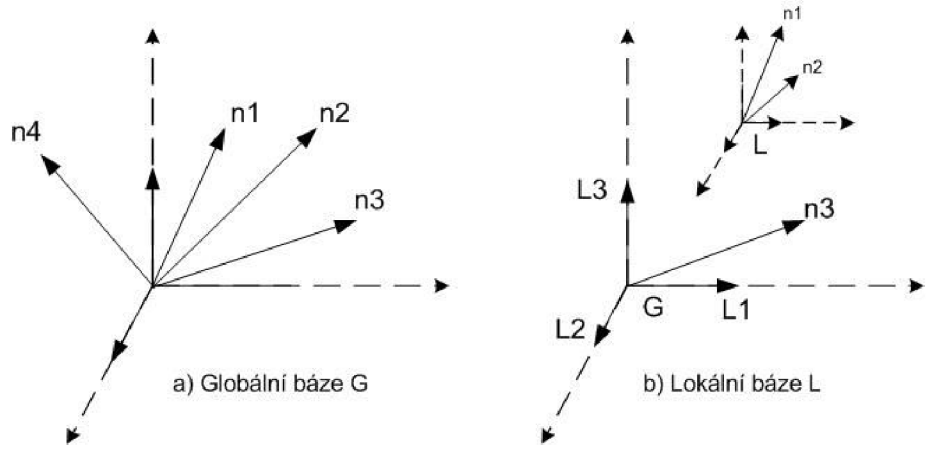
Základem celého algoritmu je přiřazení souřadnic orientačním bodům. Stanice H_i, která vstupuje do systému GNP potřebuje zjistit vzdálenost ke stanici H_j. Nejprve si tedy musí změřit vzdálenost k několika orientačním bodům O_i. Tím získá jejich souřadnice a následně pomocí optimalizační funkce zjistí své souřadnice x_j v rámci souřadnicového systému. Jestliže vyjádření souřadnic x_i a x_j v geometrickém prostoru je dostatečně přesné, lze pak snadno z geometrické vzdálenosti $|x_i, x_j|$ mezi stanicemi H_i a H_j zjistit jejich skutečnou vzdálenost, bez nutnosti dalšího měření, v rámci sítě Internet [19].

2.5.2 Algoritmus Lighthouses

Algoritmus Lighthouses je koncepcí nejblíže systému GNP. Důvodem vzniku toho algoritmu je skutečnost, že množina pevně daných orientačních bodů (tzv. landmarks) vytváří slabá místa ve fungování překryvné (logické) sítě založené na algoritmu GNP. V původním GNP algoritmu není řešeno, jak by měl systém vyplnit prázdná místa v případě nedostupnosti části orientačních bodů, tak aby se vhodně přeorganizoval. Tento nedostatek byl při návrhu algoritmu Lighthouses zohledněn a libovolná skupina serverů v systému může být pozičními severy. Každý uzel si pak může zvolit množinu pro něj referenčních serverů, které mohou být zcela odlišné od výběru ostatních uzlů. Náhodně zvolené orientační uzly se označují jako pivoty (pivots) a technika výběru pivotů jako technika pivotování (pivoting technique). Tzv. lokální báze, neboli mnohonásobné lokální

souřadnicové systémy vytváří mechanismus náhodného výběru referenčních serverů. Takto se algoritmus Lighthouse snaží zabránit vzniku slabých míst v systému, které by tak narušily jeho fungování. Lokální báze dále umožňují neomezenou škálovatelnost systému [10].

Základní princip algoritmu Lighthouse je založen na koexistenci mnohonásobné lokální báze spolu s tzv. přechodovou maticí *P*. Každý server v systému musí tuto přechodovou matici správně udržovat, a to po celou dobu své vlastní existence. Algoritmus si volí skupiny referenčních serverů, které se označují jako tzv. majáky (lighthouses). Dále si stanice za pomoci základních aritmetických a vektorových operací je schopna sama vypočítat své vlastní souřadnice v souřadnicovém modelu sítě. Souřadnice určené tímto způsobem vyjadřují relativní pozici stanice v závislosti na souřadnicích použitých majáků. Jinak řečeno jde o souřadnice vztažené ke konkrétní lokální bázi. Abychom byli schopni předpovídat zpoždění mezi servery umístěnými v odlišných lokálních bázích, byl navržen způsob výpočtu globálních souřadnic z již známých souřadnic lokání báze pomocí přechodové matice *P*. Tím, že přechodová matice *P* tvoří vazbu mezi souřadnicemi vztaženými k některé z lokálních bází a společnou globální bází souřadnicového systému, je její bezchybné udržování klíčovým prvkem. Na obrázku Obr 2.5 je znázorněn rozdíl mezi lokální bází *L* a globální bází *G* stejného souřadnicového systému [20].



Obr 2.5: *Rozdíl mezi globální bází G a lokální bází L. Obrázek vznikl na základě publikace [10].*

2.5.3 Systém PCoord

Systém PCoord se snaží zapojit stanice do vytváření co možná nejpřesnějšího geometrického modelu topologie sítě využívající peer to peer toplogii. PCoord je

decentralizovaný systém a na rozdíl od ostatních systémů stejného typu si uzly mezi sebou vyměňují zprávy tak, aby byly schopny společně sestavit co možná nejpřesnější překryvnou topologii a zahrnou do ní své sousedy. Jednotlivé stanice hledají skupinu jiných stanic právě na základě takto vytvořené topologie. Na tomto základě pak stanice určují své souřadnice [11]. Uzly vybrané PCoord stanicemi pro výpočet jejich souřadnic se nazývají waypoints, aby byly odlišeny od systému GNP, kde se jeho orientační body označují jako fixní uzly (landmarks) [21].

Existují tři různé strategie pro výběr orientačních bodů (waypoints), a to RandPCoord, CluserPCoord a ActivePCoord. RandPCoord a ClusterPCoord předpokládají, že libovolná podmnožina uzlů funguje jako zaváděcí uzel pro výpočet počátečních souřadnic. Metoda RandPCoord vybírá náhodně uzel z již existujících uzlů. Tyto uzly pak pracují jako její orientační body. Uzly u metody ClusterPCoord si na základě informací o topologii vybírají své orientační body. Tyto informace získají na základě již existujících uzlů a jejich souřadnic. Třetí a poslední metodou je metoda ActivePCoord. Tato metoda nevyužívá žádných zaváděcích uzlů. Aby jednotlivé uzly co nejvíce zpřesnily své souřadnice, procházejí interaktivním kalibračním procesem. V každém kroku tohoto interaktivního procesu si navzájem mezi sebou vyměňují zprávy. Takto společně vytvoří překryvnou síť s vlastními síťovými sousedy. Jednotlivé uzly využívají princip trojúhelníkové nerovnosti, aby nalezly své orientační body [11]. Metoda PCoord tedy obsahuje následující vlastnosti a mechanismy [19]:

- Váhovou ztrátovou funkci k rozlišení mezi uzly s menší a větší chybou a odporovým faktorem k zajištění stabilizace konvergence a zabránění oscilace.

- Mechanismus ke snížení množství uzlů pohybujících se k novým souřadnicím na základě důvěryhodného přiřazení k současné skupině ověřených souřadnic a RTT.

- Protokol, který umožňuje výměnu zpráv k rychlému objevování blízkých uzlů.

2.5.4 Algoritmus Vivaldi

Algoritmus Vivaldi využívá pro stanovení vhodných souřadnic minimalizaci chybové funkce. U algoritmu Vivaldi existuje pouze jeden typ uzlů, které na sebe navzájem působí a snaží se srovnat tak, aby celá síť vykazovala minimální dosažitelnou chybu. Jedná

se o metodu, která využívá tzv. relaxaci pružin. Tato metoda využívá pro minimalizaci chyby simulaci systému hmotných bodů $U = \{U_i\}$, propojených pružinami. Propojení stanic pružinami simuluje topologii sítě uzlů, které využívají tento algoritmus pro predikci zpoždění. Algoritmus Vivaldi přiřazuje každému spoji pružinu, jejíž klidová délka je rovná změřené vzdálenosti mezi uzly [4]. Jedná se o decentralizovaný (distribuovaný) algoritmus. Decentralizovaný, protože si nevolí žádnou pevně určenou množinu orientačních bodů, ale souřadnice jednotlivých stanic jsou vypočítány a aktualizovány na základě naměřených zpoždění v síti [5].

Pro určení souřadnic je nezbytné stanovit minimalizovanou chybovou funkci, která má v případě metody Vivaldi tento tvar:

$$E = \sum_i \sum_j (L_{ij} - \|x_i - x_j\|)^2 , \tag{2.4}$$

kde L_{ij} je zjištěné zpoždění mezi stanicemi i, j a x_i, x_j jsou souřadnice stanic i a j. Následně výraz $\|x_i - x_j\|$ udává vzdálenost uzlů i a j v daném souřadnicovém prostoru.

Minimalizace funkce je v algoritmu Vivaldi rovna součtu kvadratických odchylek z důvodů její analogie k posunutí ve fyzickém systému pružin, ve kterém je právě minimalizace energie, v síti pružin, shodná s minimalizací součtu kvadratických odchylek [5].

Funkce pro minimalizaci E je založena na centralizovaném algoritmu. Princip tohoto algoritmu spočívá ve vložení pružiny mezi každou stanici sítě *(i, j)*. Dále potom délka této pružiny v klidovém stavu je stejná jako známé zpoždění *(L_{ij})*. Potenciální energie pružiny je pak úměrná druhé mocnině posunutí z její rovnovážné pozice. Suma všech druhých mocnin posunutí je pak funkce *E*. Síla *Fij*, kterou pružina mezi uzly *i* a *j* působí na uzel *i* je definována takto:

$$F_{ij} = \left(L_{ij} - \|x_i - x_j\|\right) \times u\left(x_i - x_j\right). \tag{2.5}$$

Definice této síly vychází z Hookeova zákona. Výraz $\left(L_{ij} - \|x_i - x_j\|\right)$ určuje posunutí pružiny oproti jejímu rovnovážnému stavu a také představuje velikost síly působící na uzly *i* a *j*. Směr síly , která působí na uzel *i* představuje vektor $u\left(x_i - x_j\right)$ [5].

$$Fij = \sum_{j \neq i} Fij \,. \tag{2.6}$$

U tohoto centralizovaného algoritmu probíhá minimalizace funkce po malých časových intervalech. Souřadnice uzlu x_i jsou v každém časovém intervalu posunuty ve směru síly F_i a dále se přepočítají všechny ostatní síly. Každý časový interval je pak vyjádřen souřadnicemi uzlu *i* takto [5]:

$$x_i = x_i + F_i \times t \,, \tag{2.7}$$

kde *t* je délka intervalu. Jak daleko se v každém intervalu uzel *i* posune, určuje velikost hodnoty *t*.

2.5.1.1 Algoritmus Vivaldi s konstantním krokem

Když upravíme algoritmus na reálné podmínky, tzn. znalost pouze několika stanic a jejich vzájemnou vzdáleností, získáme tak nejjednodušší algoritmus Vivaldi. Na základě zjištěného zpoždění L_j a souřadnic již známých uzlů x_j pak každý uzel samostatně simuluje svůj pohyb v souřadnicovém prostoru. Index *i* již nebude dále používán, protože výpočet je svázán s uzlem U_i, na kterém probíhá [18].

Tyto informace však algoritmus nemusí dostávat pravidelně. Při příchodu páru (L_j, x_j) proběhne jedna iterace algoritmu. Pomocí pseudokódu lze zaspat algoritmus takto (převzato z [18]):

inicializace:
  v případě nulového vektoru *u(0)*
  $x_i = 0$

vivaldi_simple(Lj,xj):

  urči velikost chyby predikce:
  $e = L_j - \left\| x_i - x_j \right\|$
  urči směr působení síly v důsledku chyby:
  $s = u\left(x_i - x_j\right)$
  výpočet síly působící na hmotný bod:
  $F_i = e \times s$
posuv o malý krok úměrný síle:
  $x_{i+1} = x_i + F_i \times \delta$

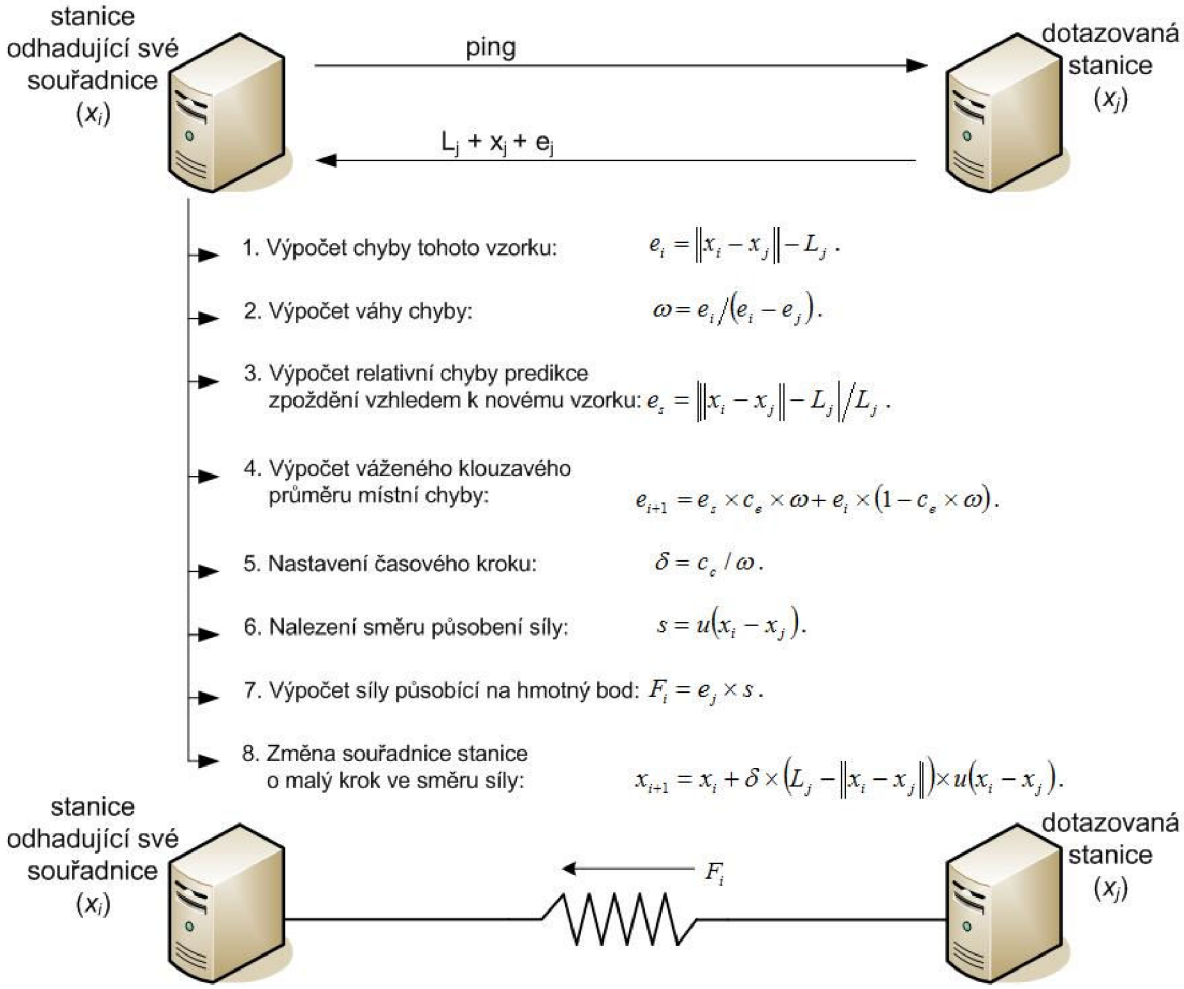
Tento zápis představuje jednu iteraci vnitřní smyčky centralizovaného algoritmu. V tomto zápise je vypočtena síla působící na bod a v závislosti úměrné na této síle (ve směru jejího působení) je jeho souřadnice posunuta. Jako časový krok je označována konstanta δ. Právě z důvodu konstantnosti tohoto časového kroku se tento algoritmus označuje jako algoritmus Vivaldi s konstantním krokem [18].

Obvykle jsou stanice na začátku umístěny v počátku souřadnicového systému. To by mohlo zapříčinit uvíznutí některých bodů, pokud by všechny jejich známé stanice měly také nulové souřadnice. Aby se tomuto předešlo musí se směr pro nulový vektor *u(0)* definovat jako náhodný jednotkový vektor [5]. Pak je tedy výsledkem odskok obou stanic na náhodný bod jednotkové kružnice se středem v počátku. V reálných podmínkách je však tento stav nepravděpodobný. Jednotlivé instance algoritmu se totiž vždy budou začleňovat do již existující sítě a budou spouštěny v různých časech [18].

2.5.1.2 Algoritmus Vivaldi s adaptivním časovým krokem

Algoritmus uvedený výše není příliš použitelný pro nasazení v praxi. Při reálných podmínkách v síti má bez dalších opatření tendenci oscilovat vlivem výskytu neustálených uzlů. Další komplikací je, že každý uzel, který se připojí do sítě může narušit stávající síť [5]. Je tedy potřeba vylepšit algoritmus tak, aby co nejrychleji nalezl své hrubé umístění a poté postupně snižoval časový krok. U algoritmu Vivaldi jsou tyto problémy řešeny jednoduchým váhováním původního časového kroku [18].

V případě, že je algoritmus Vivaldi spuštěn na síti, která předtím nebyla řazena jiným algoritmem a stanice nemají žádné souřadnice v umělém souřadnicovém prostoru, pak se jejich souřadnice určí jako počáteční (nulové). Všechny stanice tedy začínají v nulovém vektoru. Směr pro nulový vektor se nadefinuje jako náhodný jednotkový vektor a všechny stanice s nulovými souřadnicemi postupně odskočí na náhodný bod jednotkové kružnice se středem v počátku. Stejné pravidlo platí samozřejmě i pro stanice s jinými shodujícími se souřadnicemi. Tím se každé stanici určí nějaké počáteční či nekolidující souřadnice.



Obr 2.6: *Princip algoritmu Vivaldi s adaptivním časovým krokem.*

Jak je vidět na obrázku Obr 3.4, který znázorňuje pouze jednu iteraci algoritmu Vivaldi s adaptivním časovým krokem, stanice provádějící odhad svých souřadnic si nejprve zjistí pomocí nástroje *ping* vzdálenost od cílové stanice L_j. Tato cílová stanice odešle zdrojové stanici i své vlastní predikované souřadnice x_j a chybu e_j. Zdrojová stanice už také musí mít nějaké své predikované souřadnice x_i, ale ty mohou být velice nepřesné a váha jejich chyby ω se může pohybovat okolo 100 %, protože mohou být přiděleny podle nějakého náhodného klíče. Stejným způsobem stanice provádějící odhad osloví určitý počet stanic umístěných kolem ní v umělém souřadnicovém prostoru. Jak již bylo řečeno výše, pro algoritmus Vivaldi je nevhodnější 2-rozměrný euklidovský prostor s výškou. Přičemž výška zastupuje směr působení síly pružiny. Z těchto informací pak algoritmus spočítá chybu prvního vzorku e_j a hned poté určí váhu chyby ω, která vyjadřuje přesnost predikovaných souřadnic. Váha je ovlivněna jednak chybou místního uzlu e_i, dále pak chybou vzdáleného uzlu e_j. Když bude chyba vzdáleného uzlu velká v porovnání s místní chybou, pak bude jmenovatel velký, takže výsledkem je malá hodnota váhy. Daný soused

bude tedy nedůvěryhodný a uzel se posune pouze o malý časový krok. Naopak, bude-li chyba souseda malá, člen e_j bude zanedbatelný a váha se bude blížit jedné, takže se uzel poddá tlaku pružiny. Váha ω tedy představuje tuhost pružiny z Hookeova zákona, která je pro uzel tím menší, čím lépe predikuje zpoždění k druhému uzlu. Dále se provede výpočet relativní chyby odhadu e_s a váženého klouzavého průměru místní chyby e_{i+1}, kde c_c a c_e jsou ladící konstanty. Obě konstanty by měly být menší než 1 (např. $c_e = 0{,}1$ a $c_c = 0{,}9$) a slouží k nastavení časového kroku algoritmu. Po nastavení časového kroku se nalezne směr působení pružiny s a určí se síla působící na hmotný bod (na stanici) F_i . Pak již se pouze změní souřadnice stanice o malý krok ve směru síly, a tím je jedna iterace algoritmu u konce. V reálném provozu je samozřejmě zapotřebí mnoha iterací tohoto algoritmu, a to až do ustálení váženého klouzavého průměru místní chyby. Tím se pak pružiny ustálí, minimalizuje se jejich oscilace a stanice si může uložit svou predikovanou polohu spolu s její chybou. Stejným způsobem se zdrojová stanice hledající svojí polohu spojí s určitým počtem dalších uzlů a s každým z nich provádí postup popsaný výše. Aby lokalizace touto metodou byla kompletní, tak celý tento postup ještě musí zopakovat každá stanice v síti. Pak bude mít každá stanice ve zmíněné síti odhadnuty a uloženy své souřadnice v umělém souřadnicovém systému.

2.5.5 Souřadnicové systémy

Souřadnicové systémy se využívají pro vytvoření modelu síťové topologie, kterými by bylo možné předpovídat zpoždění mezi dvěma stanicemi s velkou přesností bez nutnosti provádět skutečná měření v síti. Na základě měření zpoždění mezi stanicemi je možné vyjádřit síťovou topologii pomocí daného souřadnicového systému. Všechny stanice, které chtějí být součástí takového síťového modelu, zjistí svoji síťovou vzdálenost ke skupině stanic, které jsou už součástí takového modelu pomocí měření RTT. Jakým způsobem bude měření provedeno (počet měření, výběr uzlů) je závislé na zvoleném algoritmu. Souřadnice definující pozici stanice v daném souřadnicovém systému jsou určeny na základě změřených hodnot [20].

2.5.5.1 Požadavky na souřadnicové systémy

Umělý souřadnicový systém by měl popisovat pozice stanic v Internetu s co nejmenší chybou, simulovat všechny parametry dané sítě a měl by obsahovat postačující prostor pro umístění všech stanic. Systém také musí zvládnout často velmi složité Internetové směrování, řazení do front a dobu přenosu informace datovým kanálem. Je také vhodné aby souřadnicové systémy dokázaly pracovat s velkým počtem stanic. Před centralizovanou implementací je upřednostňována decentralizovaná, protože je spolehlivější vůči výpadkům a odpovídá distribuovanému konceptu většiny aplikací. Správný umělý souřadnicový systém při komunikaci klade důraz na minimální ztížení přenosové cesty a neměl by představovat žádnou dodatečnou zátěž pro síťový provoz. Ideálně by měl systém získat všechny informace z již existující komunikace mezi aplikacemi. Dalším důležitým požadavkem je schopnost systému nastavit souřadnice svých stanic a v pravidelných intervalech se přizpůsobovat změnám v síti, které mohou nastat například vlivem přenastavení sítě, zvýšením provozu v síti či připojením nové stanice do sítě. Pokud je pro překryvnou síť zvolen nesprávný umělý souřadnicový systém, může se stát, že ani prověřený algoritmus nebude schopen předpovídat souřadnice stanic s potřebnou přesností. Z tohoto důvodu je volba správného souřadnicového systému zásadní [1].

2.5.5.2 Rozdělení souřadnicových systémů a určení počtu rozměrů

Souřadnicové systémy lze dělit na dva základní typy:

- n-rozměrné eukleidovské,
- neeukleidovské - kulové, válcové, kuželové, hyperbolické, toroidní atd.

Při testování vlastností různých algoritmů byly zkoumány různé souřadnicové systémy, nejvíce se však využívají systémy eukleidovské. Eukleidovské systémy mají podstatnou výhodu oproti ostatním systémům. Tato výhoda spočívá v tom, že systém může jednoduše navýšit počet rozměrů prostoru v případě, že při stávajícím počtu rozměrů není možné pro dané stanice nalézt řešení s dostatečně malou chybou [12]. Malý počet rozměrů neposkytuje dostatečně velký počet možností pro umístění stanic tak, aby měly malou chybu. Větší počet rozměrů zase zvedá výpočetní nároky bez zjevného zlepšení predikce. Často se také stává, že se chyba začne zvyšovat. To je zapříčiněno vlivem příliš velkého

počtu stupňů volnosti. Tento stupeň volnosti nejde běžně využívanými metodami prozkoumat, proto není možné nalézt vhodný stav s malou chybou predikce [18].

Kulové souřadnice, které jsou blízké skutečným zeměpisným souřadnicím, poskytují dostatečnou přesnost jen pro velké poloměry. Odhad se pak zpřesňuje s narůstajícím poloměrem [5]. Umělý model rozmístěných a přímo propojených stanic by pak měl malou chybu pouze pro malý interval poloměrů. To je způsobeno vysokou koncentrací stanic v Evropě, Severní Americe a v Japonsku. Tyto stanice mají mezi sebou i velmi dobrou konektivitu [13]. Při dalším rozšiřování mezikontinentálních spojů je však možné, že se topologie sítě výrazně změní a tento model pak již nebude vyhovovat. Logičtější jsou spíše, vzhledem k neobydlenosti polárních oblastí, válcové souřadnice [18].

Eukleidovský prostor

Eukleidovský souřadnicový prostor je vyjádřen touto rovností:

$$[x_1,...,x_n]-[y_1,...,y_n]=[x_1-x_n,...,y_1-y_n], \tag{2.8}$$

$$\|[x_1,...,x_n]\|=\sqrt{x_1^{\,2}+...+x_n^{\,2}}, \tag{2.9}$$

$$\alpha\times[x_1,...,x_n]=[\alpha x_1,...,\alpha x_n]. \tag{2.10}$$

U eukleidovského souřadnicového prostoru je důležitá volba jeho rozměru. V několika studiích již bylo prokázáno, že použití složitějších než 2-rozměrných, nebo 3-rozměrných souřadnic postrádá smysl. Pokud je zpoždění v síti určeno pouze zeměpisnou vzdáleností, pak 2-rozměrný souřadnicový systém tomuto případu zcela vyhovuje. Velká část zpoždění ale vzniká až v přístupové části sítě. Přidáním dalšího rozměru se však přesnost zlepšuje jen mírně, a navíc roste výpočetní náročnost a požadavky na komunikaci v síti [14].

Kulový prostor

Přesnost kulového prostoru je srovnatelná s 2-rozměrným eukleidovským prostorem. V rozsáhlejších sítích je však chybovost kulového prostoru větší. Skutečná síť „neomotává“ svoje cesty Internetem jako kolem kulového prostoru. Například u internetových cest z východní Asie bylo zjištěno, že pouze malá část komunikace do Evropy jde přímo. Většina paketů je směrována do Evropy směrem na východ. Kulovitý

model však vybírá kratší cestu, která však v tomto případě reprezentuje větší chybu při porovnání skutečného a předpokládaného RTT [15].

2-rozměrný euklidovský prostor s výškou

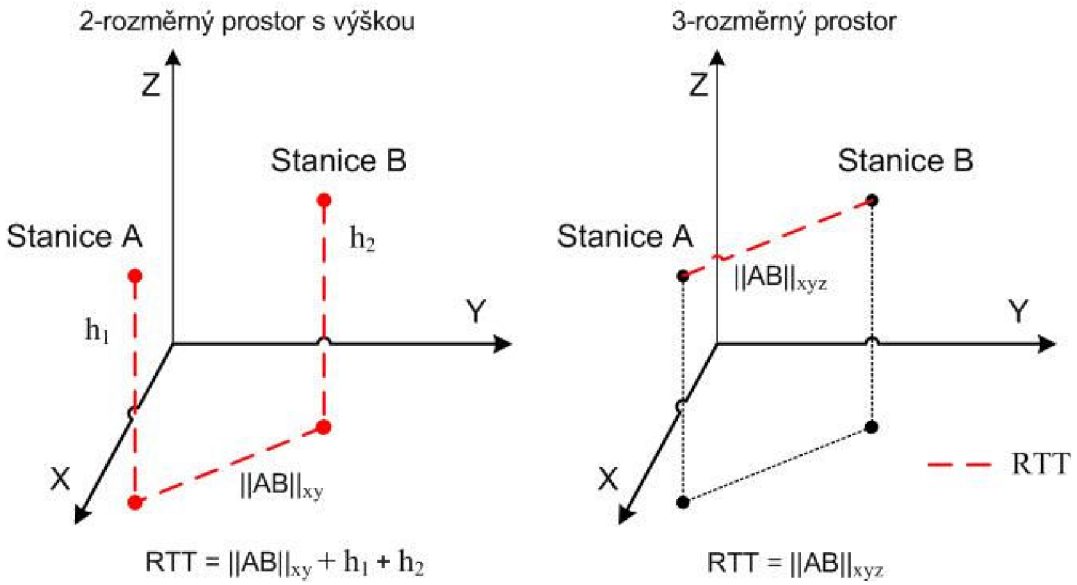
Tento 2-rozměrný eukleidovský prostor s výškou vychází z původního 2-rozměrného prostoru rozšířeného o další parametr výšku. Není to klasický 3-rozměrný prostor, protože vektorové funkce tohoto prostoru se počítají jiným způsobem (viz Obr 2.6). Operace s vektory se v tomto prostoru řídí podle následující rovnice [1]:

$$[x, x_h] - [y, y_h] = [(x - y), y_h + y_h], \tag{2.11}$$

$$\|[x, x_h]\| = \|x\| + x_h, \tag{2.12}$$

$$\alpha \times [x, x_h] = [\alpha x, \alpha x_h]. \tag{2.13}$$

Páteřní síť modelu se zpožděním úměrným k zeměpisné vzdálenosti reprezentuje eukleidovská část, kdežto výška určuje čas, který je potřeba k přenosu paketu v přístupové části sítě. Malá šířka přenosového pásma bývá nejčastěji příčinou zpoždění v této části sítě. Velikost predikovaného zpoždění je dána součtem vzdáleností výšek obou uzlů nad rovinou. Vzdálenost výšek pak zahrnuje zpoždění v přístupové síti a vzdálenost jejich cesty v eukleidovském prostoru. Toto zahrnuje hlavní rozdíl mezi n-rozměrnými eukleidovskými prostory a vektorem s výškou. Všechny stanice mají výšku ve svých souřadnicích a mohou jí podle potřeby měnit. V n-rozměrném eukleidovském prostoru se uzel, který se nachází příliš blízko dalšího uzlu od něho vzdálí, ale když má blízko kolem sebe další uzly (ze všech stran) nemá se kam posunout a začne neustále kmitat. Toto řeší právě vektor s výškou, který uzlu tlačenému ostatními uzly ze všech stran umožní upravit si vlastí výšku, a tím se síly, které na něj působí vyrovnají. Vektor s výškou dosahuje lepší přesnosti nežli 2-rozměrné a 3-rozměrné euklidovské souřadnicové systémy [1]. Z výše uvedených důvodů je 2-rozměrný eukleidovský prostor s výškou ideální pro lokalizační metodu Vivaldi, kde výška bude reprezentovat směr síly pružiny působící na stanici.



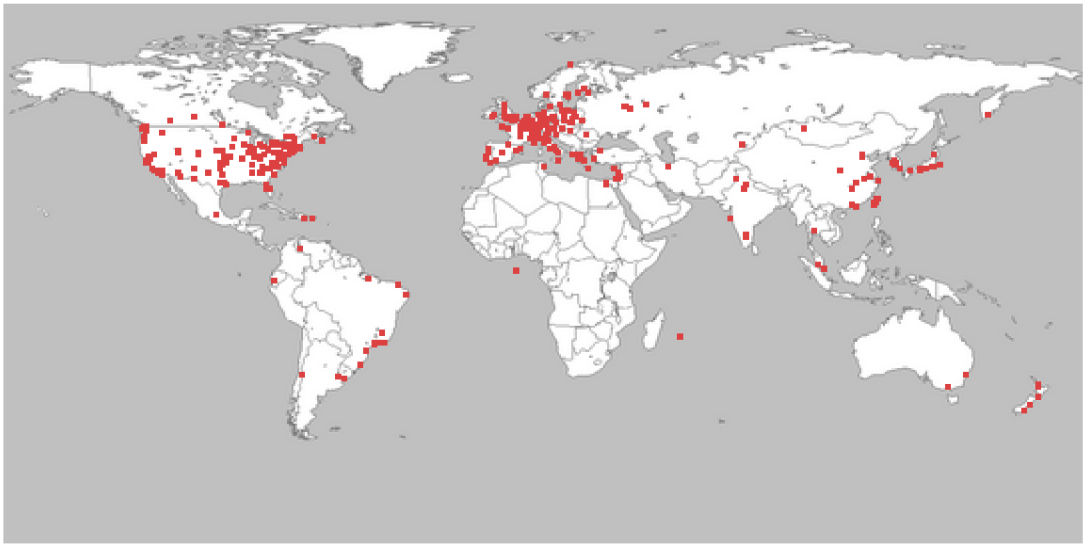
Obr 2.7: *Rozdíl mezi 2-rozměrným prostorem s výškou a 3-rozměrným prostorem. Obrázek vznikl na základě publikace [1].*

3 ZVOLENÉ PROSTŘEDÍ PRO IMPLEMETACI ALGORITMU VIVALDI

V této kapitole je popsána experimentální síť PlanetLab a operační systém Linux v distribuci CentOS.

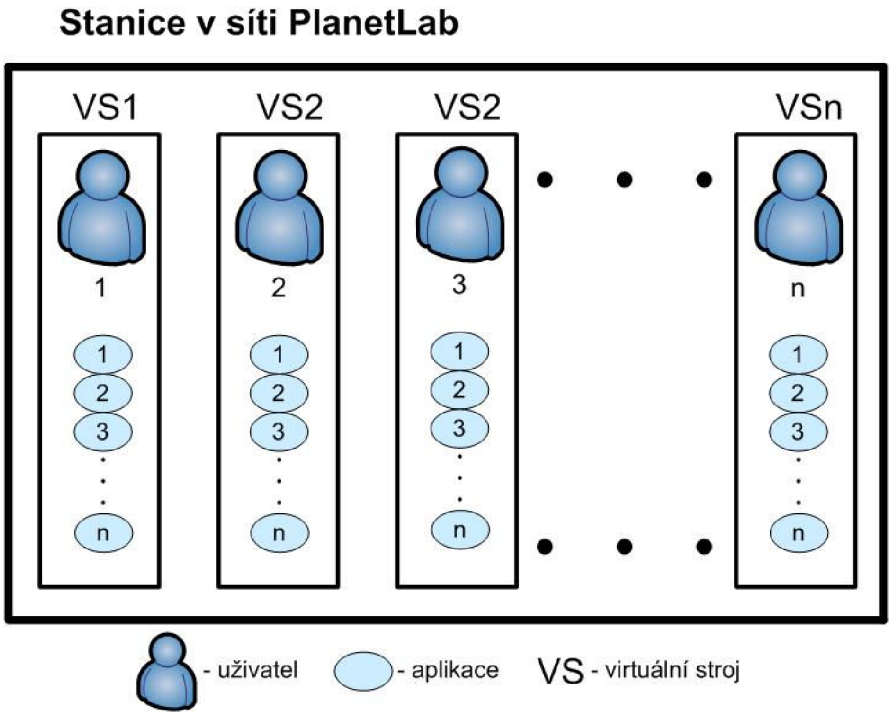
3.1 Experimentální síť PlanetLab

Experimentální síť PlanetLab vznikla v roce 2002 jako projekt několika amerických univerzit. Původně k ní bylo připojeno asi 100 stanic. Síť se nadále rozrůstala a do konce roku 2003 měla připojených již 300 stanic. V roce 2004 se do projektu zapojily i evropské university a síť planet pokračovala ve svém čilém růstu. Dnes experimentální síť PlanetLab zahrnuje již přes 1000 (přesně 1133 stanic v 517 sítích) stanic a nadále se rozšiřuje. Tyto stanice jsou rozmístěny na univerzitách, výzkumných zařízeních i v průmyslových laboratořích. Z hlediska rozvoje informačních technologií vyžívají síť PlanetLab společnosti jako např. Intel, Google, Hewlett Packard. Dále také tuto síť využívají organizace, které zajišťují síťové služby (např. provoz internetu) v akademických okruzích. Experimentální síť PlanetLab je převážně využívána k mapování sítě Internet (např. lokalizace stanic v internetu), návrhu nových komunikačních protokolů, novým postupům předávání dat pro distribuované sítě a k různým experimentálním měřením vyžadujícím reálné podmínky síťového provozu. V České republice jsou do sítě PlanetLab připojeny 3 uzly (*planetlab1.fit.vutbr.cz, planetlab1.cesnet.cz, planetlab2.cesnet.cz*), které jsou součástí českého sdružení CESNET (Czech Education and Research Network).



Obr 3.1: Mapa rozložení stanic v experimentální síti PlanetLab. *Obrázek byl převzat s publikace [30].*

Na všech stanicích v experimentální síti PlanetLab je používán operační systém Linux v distribuci CentOS (Community ENTerprise Operating System). Každý uživatel, který chce využívat tuto síť, se musí nejdříve registrovat a musí být přiřazen k určitému projektu. Pak se může vzdáleně připojit ke všem uzlům v síti. Uživatel pak má v každém uzlu vlastní virtuální stroj, který obsahuje souborový systém s omezenou velikostí a vyhrazenou výpočetní kapacitu. Díky těmto virtuálním strojům může na každé stanici pracovat více uživatelů naráz. Přičemž každý z nich pracuje právě na svém virtuálním stroji, který mu umožní spuštění i více aplikací zároveň. Samozřejmě pouze v rozsahu přidělené výpočetní kapacity a velikosti souborového systému (viz Obr 3.2).



Obr 3.2: *Stanice v síti PlanetLab a její virtuální stroje.*

3.2 Operační systém Linux - distribuce CentOS

První verze jádra Linux byla uveřejněna v roce 1991. Autor je finský student Linus Torvalds, který při návrh Linuxu vycházel z Minixu (verze optimalizovaná pro procesor 386). Jádro operačního systému Linux je vrstva mezi hardwarem a softwarem, která má na starostí zprávu paměti, přerušení a přidělování procesorového času. Jádro je rozděleno na dva základní typy. Jednak monolitické jádro, které běží v jednom paměťovém prostoru.

Dále pak mikrojádro ovládající hardware pomocí systému modulů komunikujících mezi sebou pomocí přerušení. Samotné jádro Linuxu kombinuje oba tyto koncepty [23].

Operační systém Linux nabízí mnoho instalovatelných verzí, tzv. distribucí (RedHat, SuSE, CentOS atd.). Je to dáno tím, že jádro systému a jeho základní programy (přesněji jejich zdrojové kódy) jsou volně šiřitelné a kdokoliv si na jejich základě může sestavit vlastní systém. Všechny distribuce by měly být kompatibilní s LSB (Linux Standard Base). Na ověřování kompatibility je vyvíjeno značné úsilí a je pravidelně podrobováno řadou testů. Nejaktuálnější výsledky testů jsou uvedeny na stránkách projektu LSB (*http://ldn.linuxfoundation.org/lsb/certification-applications*) [22].

RedHat je jednou s nejrozšířenějších distribucí u nás a distribuce CentOS je postavena právě na ní. Přesněji na distribuci RedHat Enterprise Linux, které je firmou RedHat poskytována v binární podobě (např. DVD) pouze platícím zákazníkům a není tedy tímto způsobem volně šiřitelná. Z licenčních důvodů jsou ale zdrojové kódy zcela volně přístupné na FTP severu, protože celá distribuce je založena na open source a free softwaru. Naproti tomu je CentOS k dispozici zcela volně, ale není provozován ani podporován firmou RedHat [24]. CentOS je dnes jednou s nejvíce využívaných distribucí na webových serverech. Některé vlastnosti distribuce CentOS:

- Aktualizace jsou vydávány se zpožděním.
- Aktualizace pomocí nástroje *yum*.
- Nemá náhradu za placené služby u distribuce RedHat.
- O zprávu balíčků se stará RPM (RPM Package Manager).
- Při kompilaci balíčků se snaží o tzv. self-hosting (distribuce překládaná sama do sebe).

4 VLASTNÍ ŘEŠENÍ

Tato kapitola obsahuje popis vytvořeného programu pro predikci zpoždění algoritmem Vivaldi s adaptivním časovým krokem a programu na přímé měření zpoždění. Dále jsou zde uvedeny použité skripty zjednodušující práci s aplikacemi a postup provedeného měření.

4.1 Realizace predikce zpoždění algoritmem Vivaldi

Do diplomové práce byla vytvořena distribuovaná aplikace Vivaldi s adaptivním časovým krokem. Tato aplikace byla realizována v programovém jazyce C++. Aplikace pracuje na reálných stanicích v experimentální síti PlanetLab, na kterých běží operační systém Linux v distribuci CentOS, proto byla tato distribuce zvolena i jako prostředí na vývoj aplikace. Tímto krokem se zabránilo možným problémům s kompatibilitou aplikace na stanicích zahrnutých v této experimentální síti.

Aplikace je založena na modelu klient-server, kde algoritmus Vivaldi běží jako klient na vybraných stanicích ze sítě PlanetLab a program *wait* pouze naslouchá na jedné vybrané monitorovací stanici s téže sítě a čeká na výstupní informace Vivaldi algoritmu. Aby bylo možné porovnat odhad algoritmu Vivaldi se skutečným zpožděním, byl realizován program na přímé měření hodnoty RTT, který pracuje téže na modelu klient-server a odesílá svůj výstup na monitorovací stanici. Využití tohoto modelu výrazně usnadní sběr výsledných dat.

Přihlášení do jednotlivých stanic v experimentální síti PlanetLab bylo realizováno pomocí SSH (Secure SHell), pro práci na jednotlivých stanicích bylo využito stejného nástroje. Na přenos aplikací a jejich následné spouštění na velkém počtu uzlů byl použit administrační program PlMan (PlanetLab experiment Manager), který je součástí softwarové podpory od provozovatelů PlanetLabu. Program automaticky po spuštění načte příslušný slice, který patří danému uživateli. Uživatel se dále může připojit k jednotlivým stanicím z přiděleného slice a uložit si tento seznam do souboru pro pozdější rychlejší přihlašování. Program dále umožňuje snadné nahrání a následné stažení souborů, provádění příkazů na každém uzlu paralelně, sledování průběhu experimentu jako celku a prohlížení výstupu uzlů na konzoly [25].

4.1.1 Popis programů

Aplikace algoritmu Vivaldi s adaptivním časovým krokem

Tato aplikace tedy pracuje jako klient na mnoha stanicích. Před samotným spuštěním programu se spustí skript *hosts.sh*, který vymaže ze souboru seznamu IP adres adresu aktuálního uzlu. Bez použití tohoto skriptu by stanice zjišťovala vzdálenost (odezvu) i k sobě, což je samozřejmě zbytečné. Dále je pak třeba zadat vstupní parametry programu (ladící konstanty, *e_jitter*, IP adresa monitorovací stanice, minimální čas běhu programu a maximální čas běhu programu). Na spuštění byl využit spouštěcí skript *ping.sh*. Oba skripty jsou popsány v kapitole 4.1.3.

Aplikace tedy po zpracování vstupních parametrů načte seznam uzlů ze zadaného souboru (*list_init()*) a inicializuje jejich souřadnice na náhodnou hodnotu. Dále pak vytvoří raw schránku pro ICMP pakety (*ping_init()*). Raw schránka je prostředek síťové komunikace umožňující přenos dat na síťové vrstvě a to je předpoklad při používání ICMP paketů, protože tyto pakety využívají právě síťové vrstvy. Dalším krokem aplikace je vytvoření UDP schránky (*udp_init()*), která slouží pro komunikaci pomocí UDP datagramů, které jsou využívány pro přenos souřadnic a chyb mezi uzly, ale také na zaslání konečných souřadnic na monitorovací stanici. Poté stanice inicializuje své souřadnice na náhodné číslo. Následně pošle *ICMP echo* (*ping_send()*) ke všem stanicím uvedeným v seznamu uzlů a čeká na odpovědi *ICMP reply* (*ping_receive()*) od dotázaných stanic. Pokud nepřijde odpověď, tak vyprší „timeout“ a ukončí se čekání i cyklus. V opačném případě algoritmus Vivaldi získá od dotazovaných uzlů odhadované souřadnice x_j s chybou vzorku e_j a vzdálenosti k daným uzlům L_j (odpovídající odezvě stanice v ms). Následně se spustí jeden krok algoritmu a po jeho proběhnutí si stanice uloží nově predikované souřadnice x_i s nově vypočtenou chybou e_i a za podmínky příchodu všech odpovědí od dotazovaných uzlů odešle tyto informace ostatním uzlům. Po tomto kroku se zkontroluje podmínka konvergence, časový limit pro běh programu a za předpokladu splnění těchto kriterií jsou zaslány výsledky měření na monitorovací stanici, kde již naslouchá program *wait* , který průběžně zpracovává příchozí informace, poté se program ukončí. Jestliže tyto kritéria splněny nejsou, algoritmus vyčká určitý časový interval a po jeho uplynutí začne znovu přijímat informace od sousedů (pokud jsou nějaké ještě vysílány), z kterých opět extrahuje souřadnice, chybu a vzdálenost. Tyto informace si přiřadí k odpovídajícím uzlům a proběhne další jeho iterace. Iterace algoritmu budou pokračovat dokud nebudou dosaženy podmínky ustálení nebo dokud nevyprší zadaný časový limit. Každá stanice si také počítá

kolik odeslala paketů (velikost v bytech) při komunikaci s jednotlivými uzly, tento údaj odesílá spolu se souřadnicemi a chybou programu *wait*.

Monitorovací program wait

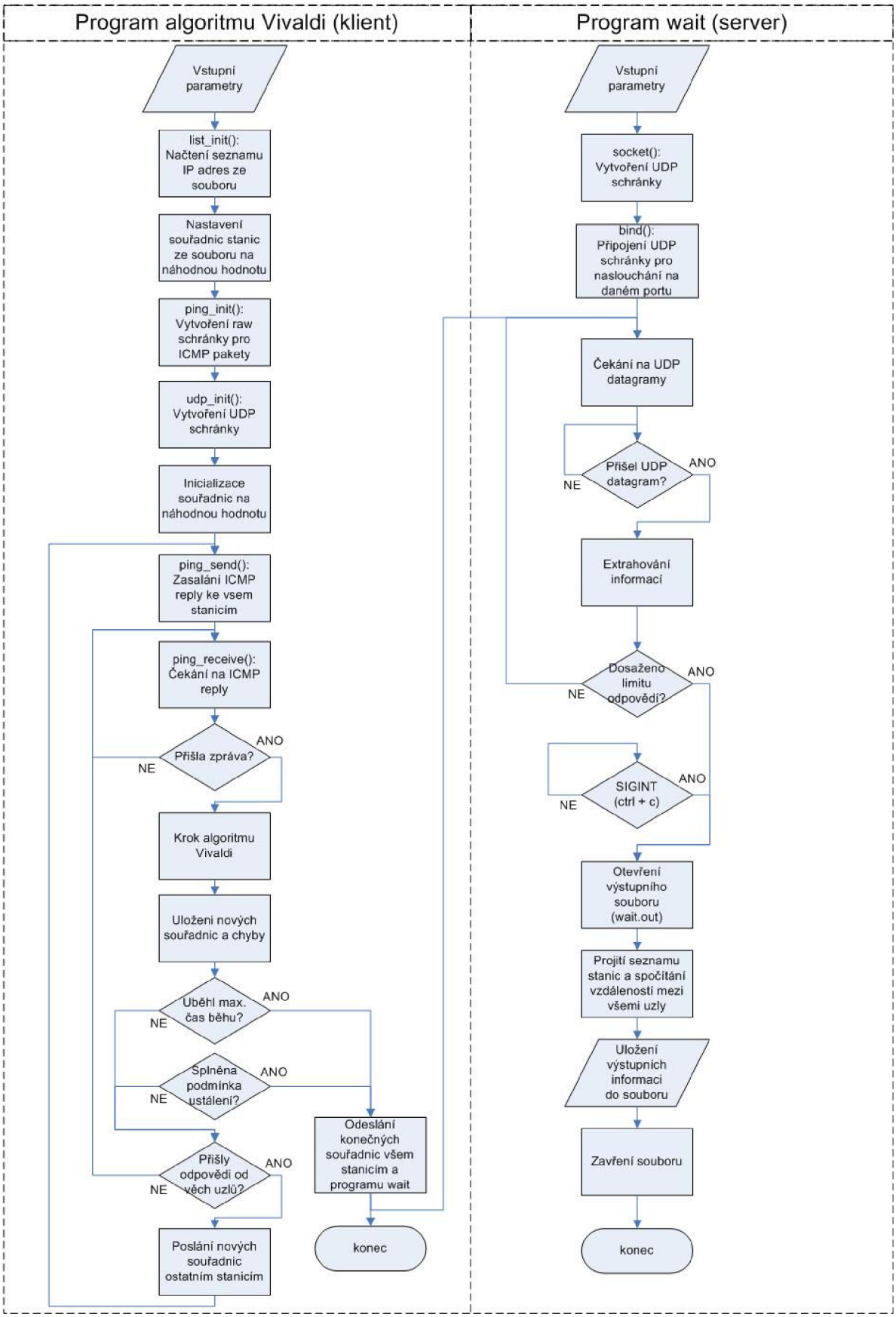
Jak již bylo řečeno výše, tento program pracuje jako server a pouze vyčkává příchozí data od měřících stanic. Program před spuštěním vyžaduje zadání vstupních parametrů (počet měření a port). Pro zjednodušení byl opět spouštěcí skript *wait.sh* (tento skript je popsán v kapitole 4.1.3), ve kterém jsou tyto parametry opět přednastaveny.

Monitorovací program *wait* nejprve alokuje seznam stanic. Nastaví obsluhu signálů *SIGINT* (*Ctrl* + *c* z klávesnice) a *SIGTERM* na funkci *sig_handler()*, která ukončí *wait* po dosažení limitu příchozích měření (systémové volání *sigaction()*). Dále vytvoří UDP schránku (systémové volání *socket()*) a připojí ji pro naslouchání na příslušném portu (systémové volání *bind()*). Poté čeká na příchozí UDP datagramy. Pokud přišel datagram, program extrahuje adresu odesílatele, souřadnice a chybu, a uloží je do seznamu stanic. Jestliže je dosaženo limitu přijatých odpovědí nebo byl program ukončen z konzole, tak otevře výstupní soubor, projde seznam stanic, pro každou stanici spočítá na základě predikovaných souřadnic vzdálenost (odezvu) ke všem ostatním stanicím a výsledek uloží do souboru *wait.out*. Program *wait* také po skončení sečte informace o zaslaných paketech (velikost v bytech), které mu byly zaslány od všech stanic a vypíše je na standardní výstup.

Aplikace pro přímé měření zpoždění

Vstupními parametry aplikace jsou číslo portu a IP adresa monitorovací stanice. Program načte řádky ze standardního výstupu (přesměrovaný a upravený výstup systémového programu *ping* ve skriptu *ping2.sh*, který je popsán v kapitole 4.1.3). Ve skriptu *ping2.sh* se dále zvolí kolikrát má proběhnout měření a soubor se seznam IP adres.

Po zadání vstupních parametrů *ping2.sh* skript provede zadaný počet měření ke všem stanicím ze seznamu IP adres a s daného počtu měření vybere nejnižší hodnotu. Po vybrání této hodnoty program *ping2* odešle UDP datagram monitorovacímu programu *wait2*, který z něj extrahuje informace a uloží je do souboru *wait2.out* (tento soubor ukládá data ve stejném formátu jako program *wait*).



Obr 4.1: *Princip programu Vivaldi a wait.*

4.1.2 Nejdůležitější funkce pro měření pomocí aplikace Vivaldi

Popis důležitých funkcí pro měření pomocí aplikace Vivaldi:

- **funkce pro vytvoření soketu:**

socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

Sokety používat pro komunikaci pomocí IP protokolu, proto je parametr hodnota makra *AF_INET*. Druhý parametr je typ soketu. Při použití UDP (User Datagram Protocol) protokolu je třeba použít makro *SOCK_DGRA*M (vytvoření datagramového soketu). Poslední parametr je protokol, který bude soketem využíván, protože zde bude použit UDP, zadáme hodnotu makra *IPPROTO_UDP* [26].

rawsock = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP)

Raw soket je prostředek síťové komunikace umožňující komunikaci na nižších vrstvách (síťová, popřípadě linková vrstva). Použití je nutné z důvodů využití ICMP protokolu, protože ten funguje na síťové vrstvě.

- **funkce sloužící pro zasílání dat daným soketem na určitou adresu a port:**

*sendto(int s, const void *msg, size_t len, int flags, struct sockaddr *to, socklen_t tolen);*

První parametr je identifikátor soketu. Druhý parametr je ukazatel na blok dat určený k odeslání. Třetím parametrem je velikost bloku dat v bitech. Dalším parametrem jsou flagy (je možno použít hodnotu 0). Pátý parametr je ukazatel na strukturu *sockaddr* který zaplníme IP adresou cílového počítače a číslem UDP portu aplikace na cílovém počítači. Poslední parametr je délka struktury, na níž se odkazuje parametr *to*. Celá funkce vrací počet skutečně odeslaných bitů, nebo v případě chyby hodnotu makra *SOCKET_ERROR* [26].

- **funkce přijme data daným soketem:**

*recvfrom(SOCKET s, char *buf, int len, int flags, struct sockaddr *from, int *fromlen);*

První parametr je identifikátor soketu. Druhý ukazatel na souvislý blok paměti, který je určen k zaplnění příchozími daty. Třetí parametr je maximální počet dat, který je do bufferu možno přijmout. Dalším parametrem jsou flagy (je možno použít hodnotu 0). Ukazatel *from* odkazuje na strukturu *sockaddr*. Ukazatel musíme předat na alokovanou strukturu, která ale nemusí být zaplněna smysluplnými daty. Parametr *fromlen* je ukazatel na číslo udávající velikost struktury, na níž se odkazuje předávaný ukazatel *from*. Po zavolání funkce *recvfrom* bude struktura na kterou se odkazuje *from* zaplněná adresou a UDP portem odesílatele dat. Číslo, na které se odkazuje *fromlen,* bude po zavolání obsahovat velikost struktury dané ukazatelem *from*. Celá funkce vrací počet bitů nebo v případě chyby *SOCKET_ERROR* [26].

- **funkce na převod binární adresy na textový tvar:**

*net_ntop(int af , const void * src , char * dst , socklen_t cnt);*

Funkce *inet_ntop* převádí binární vyjádření adresy na textové. Adresová rodina (*AF_INET*) je reprezentována v argumentu *af*. Ukazatele na buffery, ve kterých je uložena vstupní hodnota, jsou předávány argumenty *src* a *dst*. Argument *cnt* pak udává délku bufferu pro výsledek. Tato délka by měla být dostatečná pro úplné textové vyjádření dané IPv4 nebo IPv6. Funkce v případě úspěchu vrací ukazatel na *buffer*, ve kterém je uložen výsledek a v případě neúspěchu ukazatel nulový [27].

- **funkce na převod textové adresy na binární tvar:**

*inet_pton(int af , const char * src , void * dst);*

Funkce *inet_pton* převádí textové vyjádření adresy na binární. Adresová rodina (*AF_INET*) je reprezentována v argumentu *af*. Ukazatele na buffery, ve kterých je uložena vstupní hodnota, jsou předávány argumenty *src* a *dst*. Funkce v případě úspěchu konverze vrací 1, v případě špatného výstupu 0 a pokud udaná adresová rodina není známá -1 [27].

4.1.3 Práce s aplikacemi a použité skripty

Práce s aplikací algoritmu Vivaldi a použité skripty

Aplikace Vivaldi byla spouštěna skriptem *ping.sh*, který nejdříve spustí výše zmíněný skript *hosts.sh*.:

```
#!/bin/sh

/sbin/ifconfig -s | grep eth0 > /dev/null && INTERFACE=eth0
/sbin/ifconfig -s | grep eth1 > /dev/null && INTERFACE=eth1

/sbin/ifconfig $INTERFACE > /dev/null && grep -v `/sbin/ifconfig $INTERFACE | grep
'inet addr' | sed 's/\s*inet\s*addr:\([0-9.]*\).*/\1/'` < hosts > hosts.new
```

Tento skript nejdříve určí rozhraní stanice. V experimentální síti PlanetLab je to vždy *eth0* nebo *eth1* a o výběr tohoto rozhraní se postarají první dva řádky skriptu. Zbytek *hosts.sh* vybere IP adresu stanice, vymaže jí ze seznamu IP adres (v našem případě *hosts*) a vytvoří soubor s aktualizovaným seznamem IP adres (*hosts.new*).

Skript *ping.sh* dále nastaví vstupní parametry, kterými jsou ladící konstanty c_e a c_c, *e_jitter*, název souboru ze seznamem IP adres, IP adresa monitorovací stanice, číslo portu na odesílání výsledků měření, minimální a maximální čas běhu algoritmu. Příklad spouštěcího skriptu může vypadat takto.:

```
#!/bin/sh

c_e=0.1
c_c=0.9
e_jitter=0.05
IP=204.123.28.55
PORT=12345
TIME_START=0
TIME_STOP=3600

sh hosts.sh
sudo ping/ping $c_e $c_c $e_jitter hosts.new $IP $PORT $TIME_START $TIME_STOP
```

Natavení ladících konstant je důležité pro určení časového kroku algoritmu. Parametr *e_jitter* se zadává v procentech a je důležitý pro konečné ustálení algoritmu. Hodnota *e_jitter* tedy udává o kolik procent se maximálně musí lišit rozdíl několika posledních vážených klouzavých průměrů místní chyby (v našem případě rozdíl posledních pěti), k tomu aby se mohl Vivaldi algoritmus ukončit. *TIME_START* byl použit pouze na testovací účely a jen oddaluje ustálení algoritmu o daný čas, proto byl při průběhu měření

nastaven na nulovou hodnotu. *TIME_STOP* slouží jako ochrana před zacyklením programu a pouze určuje maximální možnou dobu běhu algoritmu, protože může nastat situace, která způsobí nemožnost ustálení tohoto algoritmu, jako např. kolísání odezvy některých měřených stanic, špatné nastavení ladících konstant či *e_jitter*.

Práce s monitorovacím programem wait a použitý skript

Pro program *wait* byla vybrána stabilní stanice ze sítě PlanetLab. Na tuto stanici byl poté program nakopírován pomocí PlManu a dále obsluhován přes SSH konzoly.

Monitorovací program *wait* byl spuštěn pomocí skriptu *wait.sh*. Příklad skriptu je uveden zde.:

```
#!/bin/sh

HOSTS=0
PORT=12345
wait/wait $HOSTS $PORT wait.out
```

Parametr HOSTS slouží k nastavení počtu příchozích měřeních, po kterém se může program ukončit. V našem případě je nastaven na nulovou hodnotu, která zaručuje, že se aplikace ukončí až po manuálním zásahu z konzole. Parametr PORT udává číslo portu, na kterém má *wait* naslouchat, přičemž toto číslo musí být stejné jako u Vivaldi algoritmu. Poslední parametr určuje výstupní soubor programu, kterým je *wait.out* a příklad jeho výstupu vypadá takto.:

```
122.1.115.91 128.10.19.52  241
122.1.115.91 128.112.139.18  159
122.1.115.91 128.114.63.16 144
128.10.19.52 122.1.115.91  241
128.10.19.52 128.112.139.18  52
128.10.19.52 128.114.63.16 58
```

První sloupec vyjadřuje zdrojovou IP adresu, druhý cílovou IP adresu a ve třetím je obsažena odhadovaná hodnota zpoždění mezi zdrojovou a cílovou stanicí.

Práce s aplikací pro přímé měření zpoždění a použité skripty

Program se spouští skriptem *ping2.sh*, které je popsán zde.:

```
#!/bin/sh

IP=204.123.28.55
PORT=12346
PINGS=10

sh hosts.sh

PERL='
$min = 1e10;
for ($i = 0; <>; $i++) {
  chomp;
  if ($i == 0) {
    print $_ . " ";
  } elsif ($_ < $min) {
    $min = $_;
  }
}
print $min . "\n";
'

for line in `cat hosts.new` ; do

  ping -A -c $PINGS $line | sed -n -e '/PING/p' -e '/time=/p' | grep -v 'DUP'# | \
    sed -e 's/.*time=\([^ ].*\) ms/\1/' -e 's/.*(\([^)].*)).*(.*).*/\1/' | \
    perl -e "$PERL" | ping2/ping2 $IP $PORT

done
```

První parametr *IP* slouží k nastavení IP adresy monitorovací stanice. Druhý je cílový port. Parametrem *PINGS* se nastavuje počet měření odezvy k jedné stanici. V dalším řádku dojde ke spuštění skriptu *hosts.sh*, který je popsán výše. Následující cyklus vybere nejnižší hodnotu z daného počtu měřených zpoždění. Dále se provede načtení souboru ze seznamu IP adres *hosts.new* a využije se systémového programu *ping* na zjištění odezvy mezi jednotlivými stanicemi. Naměřené hodnoty jsou pak předávány programu *ping2* a ten je odesílá na monitorovací stanici, kde naslouchá program *wait2*, který se spouští pomocí skriptu *wait2.sh*. Skript může vypadat takto.:

```
#!/bin/sh

HOSTS=0
PORT=12346

wait2/wait2 $HOSTS $PORT wait2.out
```

V tomto spouštěcí skriptu se nastavují stejné parametry jako u předchozího *wait.sh*. Port musí samozřejmě korespondovat s portem zadaným u programu *ping2*. Výstupní soubor je přednastaven jako *wait2.out* a jeho výstup je uspořádat stejně jako u souboru *wait.out*.

4.2 Postup odhadu souřadnic

Přihlašování ke stanicím v experimentální síti PlanetLab

Přihlašování k jednotlivým stanicím probíhalo pomocí SSH. Přihlášení proběhlo pomocí následujícího příkazu.:

```
ssh -l cesnet_s2 -i ~/.ssh/id_rsa planetlab2.cesnet.cz
```

Jako uživatelské jméno je zde použit název přiděleného slice *cesnet_s2*. Další parametr je privátní SSH klíč a poslední pak stanice, na kterou se chceme připojit. Dále pak bylo třeba na webové rozhraní PlanetLabu načíst veřejný klíč.

Pro práci na mnoha stanicích paralelně a na přenos programů a skriptů na více stanic, byl použit program PlMan. Stejný program byl použit i na stahování dat z daných stanic. Před samotným spuštěním PlMan se nejdříve musí nastavit SSH přístup (veřejný a privátní klíč) a heslo. Poté se již program přihlašuje k příslušnému slice automaticky. Program taky umožňuje přihlašování ke stanicím z externího souboru.

Postup měření

Nejdříve bylo třeba vybrat stabilní stanice s experimentální sítě PlanetLab. Prvním krokem bylo připojení k příslušnému slice programem PlMan, již při tomto kroku nebylo s některými uzly navázáno spojení. Aktivní uzly pak byly testovány jednoduchými konzolovými příkazy (např. *ls*, *pwd*, *ifconfig* atd.) a nahráním (stažením) testovacího souboru. Některé stanice vůbec příkazy neprovedly, jiné měly příliš velkou odezvu nebo se vůbec nepovedl nahrát testovací soubor. Na základě tohoto testování byl vytvořen seznam stabilních stanic, které se mohou použít k měření (83 stanic).

Nejdříve byly měřící aplikace pomocí programu PlMan přesunuty na zvolenou stanici, na kterou byl doinstalován překladač *gcc c++*. Na této stanici byly všechny aplikace přeloženy. Přeložené soubory byly poté staženy a následně nakopírovány na všechny stanice, které se účastnily měření. Dále pak bylo nutno všem spouštěcím souborům měřících aplikací nastavit spouštěcí práva (pomocí příkazu *chmod*). Tento postup byl

zvolen, protože na některé stanice v síti PlanetLab nebylo možné doinstalovat *gcc c++* a tak nešly aplikace zkompilovat na každém uzlu nezávisle. Další postup měření.:

- Spuštění programu *wait* na vybrané monitorovací stanici pomocí skriptu *wait.sh* a SSH konzole.

- Připojení k seznamu stabilních uzlů v experimentální síti PlanetLab (za pomoci programu PlMan).

- Spuštění programu pro predikci zpoždění algoritmem Vivaldi s adaptivním časovým krokem pomocí skriptu *ping.sh*. Příkaz *sh ping.sh* byl zadán do konzole programu PlMan a ten je rozeslal všem připojeným stanicím.

- Po zhotovení měření, nebo po zadání *ctrl + c* na konzoly se ukončí program w*ait* a vytvoří výstupní soubor *wait.out* s predikovanými hodnotami.

- Výstupní soubor *wait.out* byl stažen pomocí PlManu a hodnoty byly dále zpracovávány v programu Microsoft Excel.

Stejný postup byl použit i pro přímé měření zpoždění mezi všemi stanicemi, s tím rozdílem, že byly použity odpovídající spouštěcí skripty, tzn. *wait2.sh* a *ping2.sh* (výstupní soubor *wait2.out*).

5 DOSAŽENÉ VÝSLEDKY

V této kapitole jsou uvedeny naměřené výsledky, postup zpracování dat a porovnání lokalizační metody Vivaldi s metodou King.

5.1 Postup zpracování dat

Jelikož se pracovalo s velkým počtem stanic (83 stanic), tak bylo naměřeno velké množství hodnot. Přesněji je to 6 806 měření pro predikční metodu Vivaldi s adaptivním časovým krokem a stejné číslo pro přímé měření. To je tedy dohromady 13 612 měření a tedy i vlastně hodnot ke zpracování. S tohoto důvodu zde nebudou uvedeny všechny hodnoty, ale pouze jejich příklady.

Úkolem bylo porovnání predikovaného zpoždění metodou Vivaldi s reálným naměřeným zpožděním. Naměřené hodnoty byly uloženy v textových souborech *wait.out* (odhad Vivaldi algoritmem) a *wait2.out* (přímé měření zpoždění). Každý řádek v těchto souborech znamená jedno měření mezi zdrojovou a cílovou stanicí. Příklad obsahu obou souborů.:

```
wait.out

122.1.115.91 128.10.19.52 241
122.1.115.91 128.112.139.18 159
128.10.19.52 122.1.115.91 241
128.10.19.52 128.112.139.18 52
```

```
wait2.out

122.1.115.91 128.10.19.52 174
122.1.115.91 128.112.139.18 204
128.10.19.52 122.1.115.91 179
128.10.19.52 128.112.139.18 39
```

První sloupec je zdrojová IP adresa, druhý cílová IP adresa a třetí naměřené zpoždění. Informace s těchto souborů byly převedeny do programu Microsoft Excel a v něm pak nadále zpracovávány. Pro porovnání jednotlivých hodnot zpoždění byla vypočtena jejich relativní chyba , která je vyjádřena následujícím vzorcem:

$$\delta = \left| \frac{b-a}{b} \right| \cdot 100 \ [\%]. \tag{5.1}$$

Kde *b* (přímé měření) je skutečná hodnota a *a* (odhad Vivaldi algoritmem) je hodnota odhadovaná. Příklad tabulky v Excelu s vypočtenou relativní chybou vypadá následovně.:

Vivaldi algoritmus			**Přímé měření**			
zdrojová adresa	cílová adresa	odezva [ms]	zdrojová adresa	cílová adresa	odezva [ms]	δ [%]
122.1.115.91	128.10.19.52	241,11	122.1.115.91	128.10.19.52	174,00	38,57
122.1.115.91	128.112.139.18	158,86	122.1.115.91	128.112.139.18	204,00	22,13
122.1.115.91	128.114.63.16	144,45	122.1.115.91	128.114.63.16	120,00	20,38
122.1.115.91	128.119.41.211	295,98	122.1.115.91	128.119.41.211	183,00	61,74
122.1.115.91	128.135.164.193	233,26	122.1.115.91	128.135.164.193	160,00	45,79
122.1.115.91	128.151.65.101	117,10	122.1.115.91	128.151.65.101	176,00	33,47
128.10.19.52	133.15.59.1	262,29	128.10.19.52	133.15.59.1	215,00	21,99
128.10.19.52	133.9.81.164	180,47	128.10.19.52	133.9.81.164	196,00	7,92
128.10.19.52	134.121.64.4	90,13	128.10.19.52	134.121.64.4	91,00	0,96
128.10.19.52	134.226.52.34	157,65	128.10.19.52	134.226.52.34	151,00	4,41
128.10.19.52	137.132.80.106	275,69	128.10.19.52	137.132.80.106	269,00	2,49
128.10.19.52	137.132.80.110	275,59	128.10.19.52	137.132.80.110	263,00	4,79

Tab 5.1: *Příklad části tabulky naměřených a spočtených hodnot.*

Relativní chyba se vypočte u všech naměřených hodnot a určí se její průměrná hodna, která činí přibližně 31 %. Dále je pak třeba určit směrodatnou odchylku ze všech spočtených relativních chyb. Ta je dána vztahem:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2} \ , \tag{5.2}$$

kde N je celkový počet hodnot, x_i vyjadřuje jednotlivé hodnoty relativních chyb a $\bar{x}$ určuje jejich průměrnou hodnotu (střední hodnotu). Směrodatná odchylka tedy dosahuje asi 57 %.

Dále bylo třeba vytvořit distribuční funkci pro naměřené hodnoty. Distribuční funkce je nezáporné číslo menší nebo rovno jedné (tzn. je z intervalu $\langle 0,1 \rangle$). Funkce je neklesající zprava spojitá. Jde tedy o funkci která každému reálnému číslu přiřazuje pravděpodobnost, že náhodná veličina nabude hodnoty, která je menší nebo rovna tomu danému reálnému číslu [29]. Dala by se tedy vyjádřiv vztahem [28]:

$$F(x) = P\left[X \le x\right], \quad x \in R. \tag{5.3}$$

Jinak řečeno, pomocí distribuční funkce můžeme zapisovat pravděpodobnost, že hodnoty X padnou do intervalu $(a,b\rangle \subset R$, respektive že X nabude právě hodnoty $a \in R$. Toto tvrzení se dá vyjádřit těmito vztahy:

$$P[X \in (a,b\rangle] = F(b) - F(a), \tag{5.4}$$

$$P[X = a] = F(a) - F(a-), \tag{5.5}$$

kde $F(a-)$ značí limitu F v bodě a zleva. Každá distribuční funkce popisuje nějaké rozdělení reálných čísel a je distribuční funkcí nějaké náhodné veličiny [28].

Na základě všech relativních chyb, průměrné relativní chyby a směrodatné odchylky byla určena distribuční funkce. Na realizaci distribuční funkce byla využita funkce Excelu *NORMDIST* (viz Obr 5.1).

Distribuční funkce byla vybrána, protože při takto velkém počtu měření dokáže dostatečně vyjádřit (na základě pravděpodobnostního výskytu hodnot) přesnost tohoto měření.

5.2 Diskuze naměřených výsledků

Predikce zpoždění algoritmem Vivaldi s adaptivním časovým krokem vycházela nejlépe s ladícími konstantami c_e = 0,1, c_c = 0,9 a *e_jitter* = 0,05 (tzn. 5 %). Pro přímé měření odezvy bylo použito 10 měření ke každé stanici a s těchto deseti měření byla vybrána nejnižší hodnota, protože při zpracování dat síťovým rozhraním dochází ke zvýšení odezvy a tato hodnota nejvíce odpovídá reálnému zpoždění mezi oběma stanicemi.

Pro porovnání obou naměřených zpoždění byla vypočtena relativní chyba. Nejmenší relativní chyba vyšla 0,007 %, největší pak 520 % (viz Tab 5.2) a průměrná relativní chyba měření dosáhla hodnoty 31 %.

Vivaldi algoritmus			**Přímé měření**			
zdrojová adresa	cílová adresa	odezva [ms]	zdrojová adresa	cílová adresa	odezva [ms]	δ [%]
128.114.63.16	128.135.164.193	409,64	128.114.63.16	128.135.164.193	66,00	520,662
194.29.178.13	158.130.6.253	119,99	194.29.178.13	158.130.6.253	120,00	0,007

Tab 5.2: *Tabulka stanic s největší a nejmenší naměřenou relativní chybou.*

Přesnost naměřených zpoždění mezi jednotlivými uzly velmi kolísala. Tato skutečnost je i dobře vidět v tabulce Tab 5.1, kde se relativní chyby odezev poměrně

výrazně liší. Je to ostatně vidět i ve velkém rozdílu mezi minimální a maximální relativní chybou a také průměrnou relativní chybou. Ovšem většinou se relativní chyba měření vešla do 30 %. Toto tvrzení je vidět v grafu distribuční funkce znázorněné na obrázku Obr 5.1, kdy se křivka právě okolo relativní chyby 30 % zvedá prudce vzhůru a rychle dosahuje zmíněné maximální relativní chyby 520 %. Příklad naměřených chyb do 30 % je znázorněn v tabulce Tab 5.3.

Vivaldi algoritmus			**Přímé měření**			
zdrojová adresa	cílová adresa	odezva [ms]	zdrojová adresa	cílová adresa	odezva [ms]	δ [%]
216.48.80.12	131.123.34.35	54,86	216.48.80.12	131.123.34.35	48,00	14,29
216.48.80.12	131.179.150.72	100,73	216.48.80.12	131.179.150.72	87,00	15,78
216.48.80.12	131.193.34.21	17,75	216.48.80.12	131.193.34.21	19,00	6,55
216.48.80.12	131.246.19.201	123,96	216.48.80.12	131.246.19.201	115,00	7,79
216.48.80.12	132.239.17.224	68,61	216.48.80.12	132.239.17.224	89,00	22,91
216.48.80.12	133.15.59.1	155,06	216.48.80.12	133.15.59.1	198,00	21,68
216.48.80.12	133.9.81.164	208,19	216.48.80.12	133.9.81.164	188,00	10,74
216.48.80.12	134.121.64.4	59,42	216.48.80.12	134.121.64.4	70,00	15,11
216.48.80.12	134.226.52.34	146,69	216.48.80.12	134.226.52.34	122,00	20,24
133.9.81.164	128.220.251.50	228,40	133.9.81.164	128.220.251.50	183,00	24,81

Tab 5.3: *Příklad tabulky s relativní chybou do 30 %.*

V některých případech se hodnoty odezev lišily více jak o zmíněných 30 %. Vzorek takovéhoto měření je uveden v tabulce Tab 5.4.

Vivaldi algoritmus			**Přímé měření**			
zdrojová adresa	cílová adresa	odezva [ms]	zdrojová adresa	cílová adresa	odezva [ms]	δ [%]
128.10.19.52	128.135.164.193	111,25	128.10.19.52	128.135.164.193	21,00	429,77
128.10.19.52	128.151.65.101	57,57	128.10.19.52	128.151.65.101	43,00	33,88
128.10.19.52	128.220.251.50	52,91	128.10.19.52	128.220.251.50	33,00	60,35
128.10.19.52	128.227.56.82	88,70	128.10.19.52	128.227.56.82	28,00	216,77
128.10.19.52	128.252.19.18	115,26	128.10.19.52	128.252.19.18	38,00	203,31
128.10.19.52	128.252.19.19	48,39	128.10.19.52	128.252.19.19	17,00	184,63

Tab 5.4: *Příklad tabulky s relativní chybou přesahující 30 %.*

V obou tabulkách je vidět, že při malých odezvách a poměrně malých rozdílech mezi nimi, je relativní chyba dosti vysoká. Např. u odezev 17,75 ms a 19,00 ms je relativních chyba 6,55 % i když rozdíl těchto dvou zpoždění není příliš velký. Dalším takovým příkladem může být dvojice odezev 52,91 ms a 33,00 ms, kde relativní chyba stoupne až na 60,35 %. Posledním příkladem jsou zpoždění 48,39 ms a 17,00 ms, zde se chyba vyšplhala až na 184,63 %. Tento fakt může přesnost měření zkreslovat, ale při větších

hodnotách obou zpoždění jsou i při větších rozdílech relativní chyby poměrně malé, to je vidět například u odezev 228,40 ms a 183,00 ms, kde relativní chyba je 24,81 %. Tímto se přesnost měření do jisté míry vyrovnává.

Dalším faktorem, který mohl zapříčinit zvýšení chyby odezvy může být kolísání zpoždění jednotlivých stanic. Přímým měřením se totiž určí pouze jedna určitá hodnota zpoždění, ale algoritmus Vivaldi počítá z mnoha odezvami mezi stanicemi (než dojde k jeho ustálení) a to může mít za následek určité zkreslení.

5.2 Porovnání s lokalizační metodou King

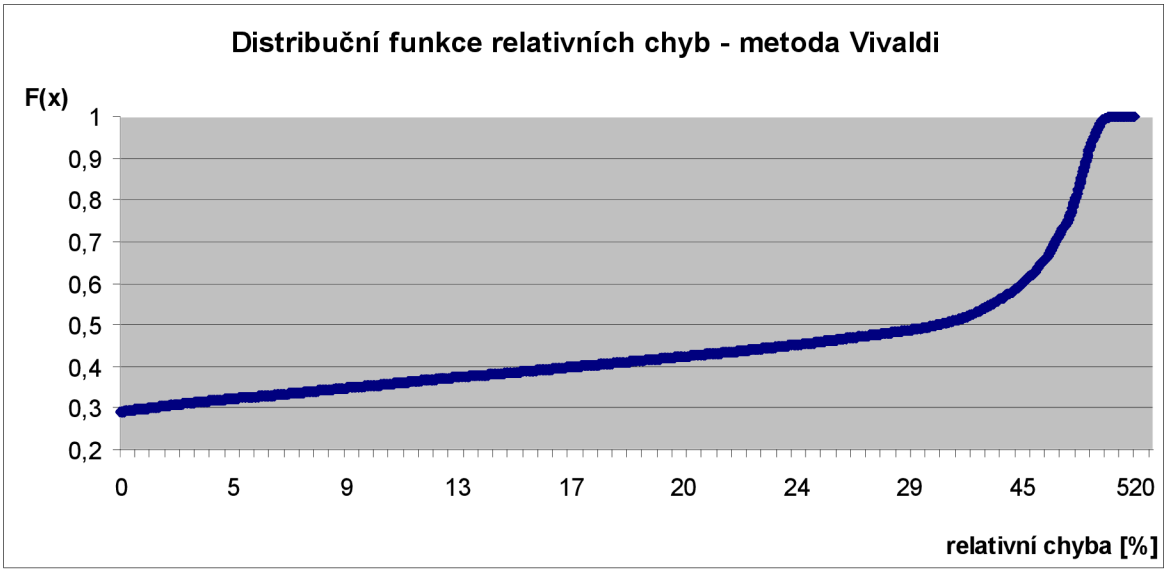
Porovnávána byla lokalizační metoda Vivaldi s adaptivním časovým krokem s lokalizační metodou King. Realizaci měření pomocí metody King provedl Bc. Michal Exler v rámci své diplomové práce. Jak již bylo řečeno výše, obě lokalizační metody byly porovnávány se skutečnými odezvami mezi jednotlivými stanicemi a na základě naměřených výsledků byly určeny relativní chyby mezi skutečnými a predikovanými zpožděními.

	min. relativní chyba [%]	max. relativní chyba [%]	průměrná relativní chyba [%]
Vivaldi	0,007	520,622	31,247
King	0,0006	482,381	23,274
rozdíl [%]	0,0064	38,241	7,973

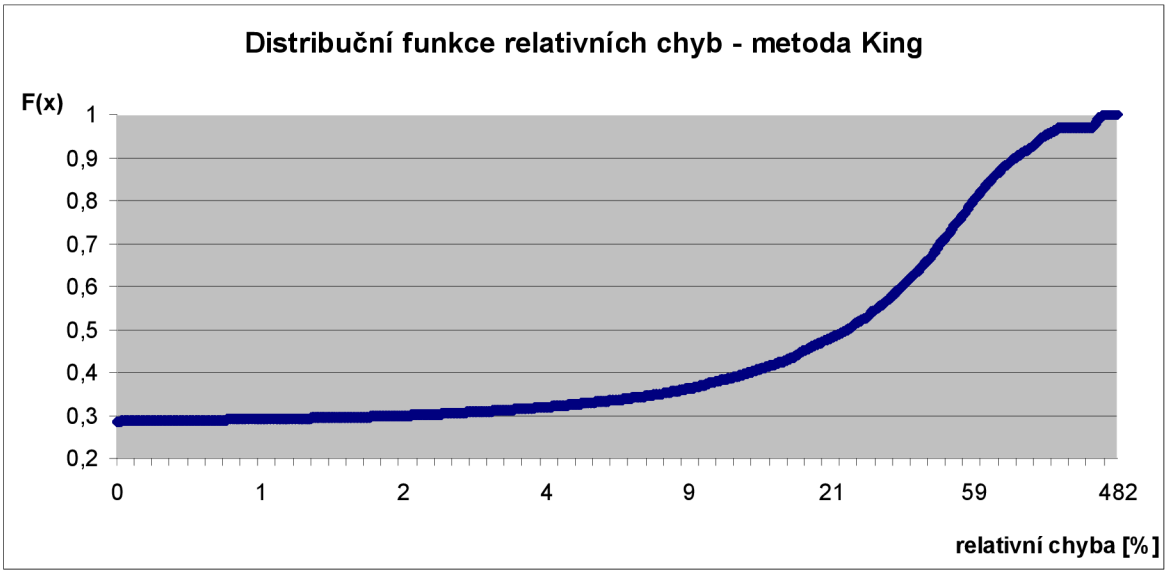
Tab 5.5: *Porovnání lokalizačních metod Vivaldi a King.*

Z tabulky Tab 5.5 je vidět, že nejnižší hodnoty relativních chyb jsou velice nízké a nedosahují ani jednoprocentní hranice. Nejvyšší hodnoty chyb obou lokalizačních metod jsou také velmi podobné a sahají k 500 %. Nejlépe o přesnosti metod ale samozřejmě informuje průměrná hodnota relativní chyby, která v případě lokalizační metody Vivaldi činí zhruba 31 % a u metody King dosahuje cca 23 %. Na základě tohoto měření je tedy metoda predikce zpoždění King přesnější o necelých 8 %.

Dále byly vytvořeny distribuční funkce relativních chyb obou lokalizačních metod a na jejich základě můžeme nadále porovnávat jejich přesnost (viz Obr 5.1 a Obr 5.2). Směrodatná odchylka u metody King je 41 %.



Obr 5.1: *Distribuční funkce relativních chyb měření pro metodu Vivaldi.*



Obr 5.2: *Distribuční funkce relativních chyb měření pro metodu King.*

Distribuční funkce lokalizační metody Vivaldi s adaptivním krokem zvolna stoupá až k 30 % relativní chyby a následně začne strmě stoupat až na maximální relativní chybu. U metody King má funkce velmi podobný průběh, s tím rozdílem, že výrazný nárůst funkce nastane již při 20 % relativní chybě. Těmito zlomy vlastně distribuční funkce kopíruje přesnost jednotlivých metod. Z grafických závislostí je čitelný rozdíl přesností obou metod, kde lokalizační metoda King má velkou pravděpodobnost výskytu velmi malých relativních chyb, kdežto metoda Vivaldi je na tom podstatně hůře. Z grafu je patrné to, že u algoritmu

Vivaldi se vyskytuje mnohem větší množství značných relativních chyb a to od 45 % do 520 %, kdežto distribuční funkce metody King jasně znázorňuje mnohem menší výskyt vysokých hodnot chyb predikce odezvy. Toto je hlavní důvod, proč odhad zpoždění pomocí této metody byl přesnější.

6 ZÁVĚR

V této diplomové práci byla rozebrána problematika lokalizace stanic v Internetu a metody její realizace. Jak již bylo uvedeno v teoretickém úvodu existují dvě hlavní skupiny těchto metod. První skupina realizuje predikci zpoždění pomocí přímého měření a nemusí tak využívat umělého souřadnicového systému a překryvných sítí. Predikce druhé skupiny těchto lokalizačních metod je založena na predikci zpoždění pomocí umělých souřadnicových prostorů s využitím překryvných sítí. Každá z těchto metod pak používá jistých matematických operací na určení souřadnic uzlů v daném souřadnicovém prostoru. Některé metody přímé predikce zpoždění, ale i metody využívajících umělých souřadnicových systémů si podle nějakého klíče volí omezený počet referenčních stanic, které pak oslovují jím blízké stanice a snaží se o lokalizaci jejich souřadnic. V takovémto případě se tyto metody dají označit jako centralizované. Avšak algoritmus Vivaldi s adaptivním časovým krokem, který je řešen v této diplomové práci, těchto referenčních uzlů nevyužívá. V tomto algoritmu se opírají uzly sami o sebe a není zde implementována žádná centralizace, jedná se tedy o čistě decentralizovaný distribuovaný algoritmus. Výhodou tohoto decentralizovaného algoritmu je, že nemůže dojít k výpadku centrálního uzlu, který by zapříčinil narušení již sestavené struktury sítě. V případě decentralizovaného algoritmu vypadne jen jedna z mnoha stanic a síť se pak může rychle a snadno opravit. Každá z těchto lokalizačních metod postupuje podle svých algoritmů a snaží se o co nejpřesnější a nejrychlejší odhad souřadnic s co nejnižšími nároky na síťovou komunikaci a výpočetní techniku. Samozřejmě každá z těchto metod vykazuje při měření různé výsledky (přesnost, rychlost, náročnost).

V rámci této diplomové práce byl vyvinut program pro odhad zpoždění při komunikaci pomocí lokalizační metody Vivaldi s adaptivním časovým krokem a program pro přímé měření odezev mezi jednotlivými stanicemi. Odhad zpoždění metodou Vivaldi byl pak porovnám s přímým měřením a lokalizační metodou King. Průměrná relativní chyba metody Vivaldi činí 31 % a u metody King chyba dosáhla hodnoty 23 %. Výsledkem tohoto měření je, že lokalizační metoda King je o 8 % přesnější nežli metoda Vivaldi s adaptivním časovým krokem. S naměřených výsledků a sestrojených distribučních funkcí je jasně vidět, že metoda Vivaldi po celé měření dosahovala větších odchylek od skutečných odezev mezi stanicemi a to i odchylek velmi výrazných. Počet těchto velkých odchylek od reálných zpoždění byl podstatně větší nežli u lokalizační metody King. To je

tedy jeden z hlavních důvodů, které zapříčiňují větší přesnost predikce zpoždění metodou King.

Seznam použité literatury

[1] HANDL, T. *Algoritmus Vivaldi pro nalezení pozice stanice v Internetu*. Brno: Vysoké Učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2009. 69 s. Vedoucí diplomové práce doc. Ing. Dan Komosný, Ph.D.

[2] POSTEL, J., RFC793 - *Transmission Control Protocol* [online].: USC/Information Sciences Institute, září 1981. Dostupné z: <http://www.faqs.org/rfcs/rfc793.html> [cit. 4. 12. 2010].

[3] 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.

[4] COX, R. - DABEK, F. *Learning Euclidean coordinates for Internet hosts* [online]. Cambridge (USA) : Massachusetts Institute of Technology, 2002. Dostupné z: <http://www.txtr.com/text/ak489> [cit. 7. 12. 2010].

[5] DABEK, F. et al. Vivaldi: *A Decentralized Network Coordinate System. In Proceedings of SIGCOMM 2004.* [online]. Portland: MassachusettsInstitute of Technology, 2004. Dostupné z: <http://pdos.csail.mit.edu/papers/vivaldi:sigcomm/paper.pdf> [cit.7. 12. 2010].

[6] EUGENE NG, T. S. - ZHANG, Hui. *Towards Global Network Positioning* [online]. Carnegie Mellon University : Pittsburgh, PA 15213, 2000. Dostupné z: <http://conferences.sigcomm.org/imc/2001/imw2001papers/74.pdf> [cit. 8.12. 2010].

[7] COX, Russ - DABEK, Frank. *Learning Euclidean Coordinates for Internet Hosts* [online]. Cambridge (USA) : Massachusetts Institute of Technology, 2002. Dostupné z: <http://pdos.csail.mit.edu/~rsc/6867.pdf> [cit. 8. 12. 2010].

[8] NELDER, J.A. - MEAD, R. *A simplex method for function minimization* [online]. Churchill College : Cambridge, 1999. Dostupné z: <http://www.rupley.com/~jar/Rupley/Code/src/simplex/nelder-mead-simplex.pdf> [cit. 8. 12. 2010].

[9] FAN, R. *Internet coordinate system* [online]. Dostupné z: <http://pact518.hit.edu.cn/attachments/download/Internet%20coordinates> [cit. 8. 12. 2010].

[10] PIAS, M. - CROWCROFT, J.; WILBUR, S.; HARRIS, T.; BHATTI, S. *Li-ghthouses for Scalable Distributed Location* [online]. In Proc. of Second International Workshop on Peer-to-Peer Systems (IPTPS' 03). University of Cambridge, 2003. Dostupné z: <http://research.microsoft.com/users/Cambridge/tharris/papers/2003-iptps.pdf> [cit. 8. 12. 2010].

[11] LI-WEI, L. - LERMAN, S. PCoord: *Network position estimation using peer-topeer measurments* [online]. Cambridge (USA): Massachusetts Institute of Technology, Dept. of Civil and Environmental Engineering, 2005. Dostupné z: <http://web.mit.edu/lilehman/www/paper/pcoordnca04.pdf> [cit. 8. 12. 2010].

[12] SPENCE, D. *An implementation of a Coordinate based Location System* [online] Cambrigde (UK): University of Cambridge, 2003. Dostupné z: <http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-576.pdf> [cit. 8. 12. 2010].

[13] 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. Dostupné z: <http://www.cs.cornell.edu/~abrahao/docs/imc08.pdf> [cit. 8. 12. 2010]

[14] COSTA, M. - ASTRO, M. - ROWNSTORM, A. - KEY, P. PIC: *Practical Internet coordinates for distance estimation*. In *International Konference on Distributed Systeme,* Tokyo, Japan, March 2004. Dostupné z: <http://www.cl.cam.ac.uk/Teaching/2003/ AdvSysTop/pic.pdf> [cit. 9. 12. 2010].

[15] SHAVITT, Y. - TANKEL, T. *Big-bang simulation for embedding network distances in Euclidean space*. In Proc. of IEEE Infocom, April 2003.

[16] FRANCIS, P. IDMaps: *A Global Internet Host Distance Estimation Service* [online]. In IEEE/ACM Trans. on Networking [online], New York: [s.n], říjen 2001. Dostupné z: <http://www.eng.tau.ac.il/~shavitt/pub/ToN01IDMaps.pdf> [cit. 9. 12. 2010].

[17] CHEN, Y. - KATZ, H. R. - OVERTON, K. *Internet Iso-bar: A scalable overlay distance monitoring system* [online]. Dostupné z: <http://sahara.cs.berkeley.edu/papers/YCCO02.ps> [cit. 9. 12. 2010].

[18] ŠVÉDA, J. *Nalezení pozice stanic v Internetu pomocí uměle vytvořených souřadnicových systémů*. Brno: Vysoké Učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2007. Vedoucí diplomové práce doc. Ing. Dan Komosný, Ph.D.

[19] KOPEČEK, T. *Nalezení fyzické pozice stanice v síti Internet*. Brno: Vysoké Učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. Vedoucí diplomové práce doc. Ing. Dan Komosný, Ph.D.

[20] ŠIMÁK, J. *Měření vzdáleností mezi stanicemi v IP sítích*. Brno: Vysoké Učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. Vedoucí diplomové práce doc. Ing. Dan Komosný, Ph.D.

[21] ŠKVOR, M. *Lokalizace stanic v síti Internet pomocí umělých souřadnicových systémů*. Brno: Vysoké Učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. Vedoucí diplomové práce doc. Ing. Dan Komosný, Ph.D.

[22] GELNER, R. - KASPRZAK, J. *Distribuce Linuxu* [online]. Dostupné z: <http://www.linux.cz/distribuce.html> [cit. 14. 12. 2010].

[23] BEZDĚK, T. *Jádro Linux - popis konfigurace* [online]. Dostupné z: <http://www.fi.muni.cz/~kas/p090/referaty/2007-podzim/ct/kernel.html> [cit. 14. 12. 2010].

[24] www.centos.org - *About CentOS* [online]. Dostupné z: <http://www.centos.org/modules/tinycontent/index.php?id=2> [cit. 14. 12. 2010].

[25] www.cs.washington.edu - *PlanetLab experiment manager* [online]. Dostupné z: <http://www.cs.washington.edu/research/networking/cplane/> [cit. 18. 5. 2011].

[26] Dostál, R.; *Protokol UDP 1.část* [online]. Dostupné z: <http://www.builder.cz/tiraz.php?uid=275> [cit. 18.5. 2011]

[27] www.ipv6.cz - *BSD sockets - Konverze adresových formátů* [online]. Dostupné z: <https://www.ipv6.cz/BSD_sockets_-_Konverze_adresových_formátů> [cit. 8. 5. 2011].

[28] Friesl, M.; *Pravděpodobnost a statistika hypertextově* [online]. Dostupné z: <http://home.zcu.cz/~friesl/hpsb/dist.html> [cit. 18.5. 2011]

[29] Řezanková, H. - Marek, L. - Vrabec, M.; *IASTAT - INTERAKTIVNÍ UČEBNICE STATISTIKY* [online]. Dostupné z: <http://iastat.vse.cz/> [cit. 18.5. 2011]

[30] www.planet-lab.org - *PlanetLab* [online]. Dostupné z: <http:// www.planet-lab.org/> [cit. 18.5. 2011]