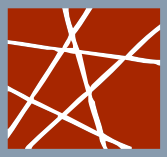

Network Architectures
and Services
NET 2012-05-1

Dissertation

Latency Prediction for P2P Overlays

Benedikt Bruno Martin Elser

Network Architectures and Services
Department of Computer Science
Technische Universität München

TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Netzwerkarchitekturen

Latency Prediction for P2P Overlays

Benedikt Bruno Martin Elser

Vollständiger Abdruck der von der Fakultät für Informatik
der Technischen Universität München
zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:		Univ.-Prof. Dr. J. Schlichter
Prüfer der Dissertation:	1.	Univ.-Prof. Dr. G. Carle
	2.	Prof. Y. Shavitt, Tel Aviv University / Israel
	3.	TUM Junior Fellow Dr. Th. Fuhrmann

Die Dissertation wurde am 20.12.2011 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 03.05.2012 angenommen.

Cataloging-in-Publication Data
Benedikt Bruno Martin Elser
Latency Prediction for P2P Overlays
Dissertation, May 2012
Network Architectures and Services, Department of Computer Science
Technische Universität München

ISBN: 3-937201-29-7
ISSN: 1868-2634 (print)
ISSN: 1868-2642 (electronic)
DOI: 10.2313/NET-2012-05-1
Network Architectures und Services NET 2012-05-1
Series Editor: Georg Carle, Technische Universität München, Germany

Abstract

Peer-to-peer (P2P) applications have become countless since they were initially proposed about ten years ago. They became popular among the general public when being used as a platform for file sharing. However, their robustness and scalability sparked interest from industry and research. Today these concepts drive some of the main businesses as back ends or migrate intensive workload away from them. Recently, Microsoft added P2P capabilities to their update mechanism. Adobe provides P2P APIs in their Flash product. Amazon's Dynamo and Skype are two examples of commercial success in this area. One popular non-commercial example is the BitTorrent protocol.

P2P is a movement "back to the roots" of the Internet. The early ARPANET was a system of equal computers providing services to each other. However, since that time the core assumptions have changed. In the first place, the Internet has changed dramatically in size. With the introduction of commercial Internet Service Providers in 1989 it became apparent that the Internet's participants were no longer equally capable: Dedicated bandwidth-rich machines provided services to customer machines, connected via narrow-band dial-up links. Henceforth, the original P2P architecture was no longer applicable and the classical Client/Server architecture became the predominant model.

With the rapid growth of the Internet even Client/Server suffered scalability issues. At the same time the consumer Internet access saw huge changes. Persistent dial-up connections became common along with an increase of the offered bandwidth. Furthermore, academia introduced resilient algorithms that finally allowed the self-organisation of a large number of hosts. Hence, a "Renaissance" of the P2P paradigm began.

Today Client/Server is still the predominant paradigm. Meanwhile, P2P has proven its capabilities and found adoption. Today's P2P systems often avoid fully decentralized solutions and rather employ a hybrid approach. They employ central components to facilitate tasks such as a user management or billing. This central point creates a single point of failure, as illustrated during the recent Skype downtime of 2010. It may also constitute a bottleneck. Hence, today's systems often use federated systems, for example, in cloud computing data centers.

Despite P2P's success, there exist only few software frameworks that support the development of truly distributed applications, such as FreePastry. Others such as OpenDHT are not maintained anymore. Often the P2P technique is tied to monolithic applications, such as the eMule client and is thus not reusable.

A goal of this thesis is to introduce a software ecosystem that can foster the development of P2P software. The "IGOR ecosystem" consequently pursues the Key Based Routing (KBR) / application split, where the choice of an overlay is separated from the higher application logic e. g. a DHT implementation. This allows developers to choose an overlay architecture. It also eases maintainability. Furthermore, it is the first library that enables complex endpoint operation. Rhea et al. considered this as a missing link hampering the development of P2P applications in 2005, but to the best of our knowledge no one has addressed Rhea's concern. Another beneficial consequence from the KBR split it is the ability to simulate the software stack at two levels, thereby allowing developers to do high performance simulations of their applications by skipping one part of the stack.

The optimal overlay topology is still an open research question. In an ideal scenario clients attach preferably to nearby peers and thereby create efficient overlay structures. However, clients would need to do *a priori* measurements to identify their nearby clients. Network coordinates are an important technique that obsoletes a priori measurements. They allow the peers to predict to which other peers they are likely to experience a low round trip time.

Traffic in the Internet is known to change over time, e. g. it exhibits volatile behaviour. One major challenge for a network coordinate algorithm is its ability to handle these fluctuations. The most widespread algorithm is the Vivaldi algorithm. Vivaldi embeds latencies and achieves highly precise network coordinates. However, recent studies have raised doubt in Vivaldi's ability to resolve the structure of the underlying network better than at a continental level. Hence, it would fail to give a decent prediction.

In the present thesis I present an in-depth study of the Vivaldi algorithm and propose an enhancement that leads to an order of magnitude improved accuracy. I start with a simulation study from which I derive a profound understanding of the Vivaldi embedding, which was fundamental for the development of my new algorithm. I conduct the analysis using the IGOR ecosystem. To ensure realistic simulation settings I use five different established sources of static RTT data, provided by independent scientific research, as well as dynamic traces based on PlanetLab measurements that I conducted myself. The simulations focus on Vivaldi's performance and explore the algorithm's sensitivity to parameters. Hereupon I introduce the Hierarchical Vivaldi algorithm, a Vivaldi variant that leads to an order of magnitude improved accuracy. I assure its improved performance in comparative simulation studies. Hierarchical Vivaldi has been included into the popular Vuze P2P client, which allowed me to confirm its performance in a large scale real world setting. Over seven million users ran the software, over 2000 took part in our evaluation study, contributing extended measurements. The results from that planet scale evaluation and a brief summary of our raw measurement data for the scientific community concludes this thesis.

Zusammenfassung

Seit dem Erscheinen der ersten Peer-to-Peer (P2P) Anwendungen vor mehr als 10 Jahren, erfreuen sich diese immer größerer Beliebtheit. Anfangs nur als komfortable Möglichkeit zum Dateiaustausch genutzt, zog die Robustheit und Skalierbarkeit des P2P-Konzepts schnell die Aufmerksamkeit von Wissenschaftlern und der Software-Industrie gleichermaßen auf sich. Heute stellt P2P die Technologie für einige der bekanntesten Produkte oder trägt durch das Konzept, die Arbeitslast auf teilnehmende Rechner zu verteilen, dazu bei. So erweiterte Microsoft vor kurzem den "Windows Update" Mechanismus um P2P Fähigkeiten. Adobe stellt Flash Entwicklern P2P APIs zur Verfügung. Im kommerziellen Bereich weisen Amazons Cloud-Storage Lösung "Dynamo" oder das weitverbreitete Videotelefonie-Programm Skype signifikant hohe Nutzerzahlen auf. Eine verbreitete nichtkommerzielle Anwendung ist BitTorrent.

Im Prinzip ist P2P eine Rückkehr zu den Wurzeln des Internets. Im frühen ARPANET war jeder Rechner eine gleichberechtigte Entität und stellte für alle anderen Rechner Dienste zur Verfügung. Seit dieser Zeit haben sich jedoch die Anforderungen an die Art der Diensterbringung deutlich geändert. Allein das Ausmaß des Internets hat dramatisch zugenommen. Nachdem 1989 mit den ersten kommerziellen Internet Providern das Netz Privatleuten zugänglich gemacht wurde, bestand das Internet nicht mehr nur aus gleichwertigen Diensterbringern. Dedizierte Maschinen, ausgestattet mit enromen Kapazitäten an Arbeitsspeicher, Rechenleistung und zur Verfügung stehenden Bandbreite standen den spärlich über Modem angebundenen Rechnern von Verbrauchern gegenüber. Die klassische Client-Server Architektur mit dedizierten Dienstberbringern, hatte die Peer-to-Peer Architektur verdrängt.

Die rasante Durchdringung der Gesellschaft durch das Internet und das damit verbundene Wachstum desselben, offenbarete jedoch Engpässe hinsichtlich der Skalierbarkeit in der Client-Server Architektur. Zur gleichen Zeit ergaben sich weitreichende Änderungen auf der Seite der Endbenutzer. Die Einführung der DSL Anschlüsse mit permanenter Internetverbindung erreichte den Massenmarkt. Lösungen zur Selbstorganisation einer großen Anzahl von Rechnern fanden den Weg von der Wissenschaft in endbenutzerfreundliche Software. Die Renaissance des P2P-Paradigmas begann.

Noch immer ist zwar die Client-Server Architektur dominant. Jedoch hat auch P2P seine Leistungsfähigkeit bewiesen und ist aus der heutigen technologischen Landschaft nicht mehr weg zu denken. Heutige P2P-Systeme sind oft hybrid. Abläufe, die im verteilten Szenarien oftmals schwer zu realisieren sind, wie zum Beispiel die Authentifizierung

von Benutzern, werden von einer zentralen Komponente, etwa einer zentralen Benutzerverwaltung, erledigt. Diese hybride Architektur ist im P2P-Bereich vorherrschend. Zur Verdeutlichung: Jede der einleitend erwähnten Applikationen verwendet den hybriden Ansatz. Aber es bestehen weiter Saklierbarkeitsprobleme in hybriden Systemen, wie die fehlende Ausfallsicherheit. Der zweimalige Ausfall des Skype-Netzwerkes im Jahre 2010 sei als Beispiel angeführt. Zentrale Komponenten sind auch ein Engpass, der die Skalierbarkeit von Systemen begrenzt. Daher werden heute vermehrt massiv gebündelte Ressourcen, sogenannte "Cloud´´ Rechenzentren verwendet.

Trotz des Erfolges des P2P-Konzepts gibt es auch heute noch wenig grundlegende Software (*"Framework"*), die eine Entwicklung von völlig dezentralen Programmen unterstützt. Eine Ausnahme ist FreePastry, wohingegen OpenDHT nicht mehr aktiv gepflegt wird. Wenn überhaupt, sind P2P-Komponenten applikationsspezifisch und nicht wiederverwendbar, wie im Fall der eMule P2P-Software.

Zielsetzung der vorliegenden Arbeit ist die Vorstellung ein solches Framework, das geeignet ist, die Entwicklung vollständig dezentraler P2P-Software voranzutreiben. Das sogenannte "IGOR framework´´ setzt die Idee der Trennung einer KBR-Schicht von einer Applikationsschicht konsequent um. Die damit gewonnene Entkoppelung befreit den Entwickler von der aufwändigen und fehleranfälligen Erstellung monolytischer Insellösungen und verspricht eine komfortablere Wartung. Es entkoppelt darüberhinaus die Netzwerk-Abstraktion (das *Overlay*) von höher angesiedelten Schichten, wie einer verteilten Hash-Tabelle (DHT). Darüber hinaus, ist es zeitlich gesehen die erste Lösung, die konsequent komplexe Operationen an Endpunkten ermöglicht. Derartige Operationen wurden bereits 2005 von S. Rhea gefordert, da sie die Verbreitung von P2P unnötig behindern. Die Unterscheidung zwischen Overlay und darüber liegender Schicht hat des weiteren den Vorteil, daß Komponenten getrennt simuliert werden können und besonderes Augenmerk auf den jeweils zu simulierenden Teil einer Applikation gelegt werden kann.

Eine der vordringlichsten Herausforderungen in P2P-Umgebungen ist die Zuteilung von Ressourcen unter den Teilnehmern. Wie kann ein optimales Overlay aufgespannt werden? Im Idealfall würden sich die Teilnehmer selbst mit möglichst Latenz-nahen Teilnehmern verbinden. Wenn dieser Prozess von allen Teilnehmern ausgeführt wird, entsteht ein effizientes Overlay. Um diese nahen Teilnehmer zu finden, müsste jeder Teilnehmer *a priori* Messungen durchführen. Mit Netzwerk-Koordinaten steht jedoch ein dedizierter Algorithmus zur Verfügung, der Messungen unnötig macht. Der Algorithmus erlaubt Teilnehmern, die zu erwartende Latenz untereinander vorherzusagen. Damit werden die Nachbarn herausgefunden, mit denen eine geringe Latenz zu erwarten ist.

Latenzen sind im Internet zeitlichen Änderungen und stark schwankendem Verhalten unterworfen. Ein Qualitätsmaß für einen Netzwerk-Koordinaten-Algorithmus ist

dessen Fähigkeit, mit diesem Verhalten umzugehen. Einer der verbreitetsten Netzwerk-Koordinaten-Algorithmen, der diese Ansprüche erfüllt, ist Vivaldi. Dieser Algorithmus nimmt eine Latenz-Einbettung vor, die sehr präzise Netzwerk-Koordinaten erstellt. Jedoch finden sich in der Literatur der letzten Jahre Zweifel an Vivaldis Fähigkeit, Strukturen genauer als bis zu einem interkontinentalen Mass aufzulösen. Damit würde der Algorithmus verfehlen, eine verwertbare Vorhersage zu treffen.

Der Beitrag der vorliegenden Arbeit ist unter anderem eine grundlegende Analyse des Vivaldi-Algorithmus. Der Schwerpunkt dieser Arbeit besteht in der Einführung einer hierarchischen Variante des Vivaldi-Algorithmus, die zu einer deutlichen Verbesserung der Latenz-Vorhersage führt. Dieser Schritt setzt zunächst eine intensive Analyse des Vivaldi-Algorithmus voraus. Für die Analyse wird die zuvor entwickelte Software Architektur "IGOR" und ihre Komponenten verwendet. In Simulationen präsentiert die vorliegende Arbeit dem Leser die Effizienz von Vivaldi. Darüberhinaus wird die Auswirkung der c_c und c_e-Parameter auf den Algorithmus erforscht. Die Analyse bestätigt frühere Erkenntnisse. Andererseits wurden oftmals in den original Publikationen zu hohe Parameter Werte empfohlen. Darauf aufbauend stellt die vorliegende Arbeit den hierarchischen Vivaldi-Algorithmus vor, der im Rahmen dieser Arbeit entwickelt wurde. Durch die hierarchische Strukturierung der Latenz-Einbettungen wurde eine substantiell verbesserte Vorhersageleistung erreicht. Diese Ergebnisse wurden in Simulationen und in einer Kontinente übergreifenden Messung verifiziert. Bei dieser Messung wurde der Algorithmus auf über 6 Millionen Rechnern ausgeführt. Den Abschluss der Arbeit bildet die Publikation der erhobenen Messdaten zur weiteren Analyse durch die Forschugsgemeinschaft.

Contents

1. Introduction

artography is an ancient art that is older than Roman or Greek civilisations. At all times, mankind sought to build a graphical representation of their surroundings. Over time the art of producing such maps improved: In ancient Greece the first maps based on mathematical and geometrical models were developed. Furthermore, coordinates as means to express positions in a metric space appeared. Due to the intensive application in nautical navigation in the 12^{th} century the accuracy of maps was significantly improved. Today we have different elaborate kinds of maps and various coordinate systems. For example, it is easily possible to measure the distance from Munich to Rome with the help of a map using both cities' coordinates, without having to actually travel that distance. The quality of a map is measured in terms of accuracy, degree of completeness or readability.

However, this kind of information — in this thesis I focus on latency information — is not available for the Internet. There are illustrations, depicted as "Internet maps", but their practical utility is limited. Some of these maps use Geo location data of autonomous systems, some use GPS data of contributing hosts. None of these "Internet maps" has a real value, as to measure distances in terms of latency. There is no projection that maps two IP addresses to their mutual latency, because latency data is not metric.

It is, however, a desirable property to deduce the latency between two computers easily. For example, Google optimizes their products for latencies as delays impact business metrics: Answering requests from a nearby server decreases the time a user has to wait, in other words improves the user experience. The result of a bad user experience is the loss of customers. Therefore, a sizable effort is made to minimize latencies using a variety of techniques, such as redirections using the Domain Name System.

A technique to predict and thus being able to minimize latencies is to assign coordinates to nodes. Just as the process of deducing the geometric distance between cities a node's coordinate can be related to other coordinates to deduce latency. This technique doesn't involve any previous communication between two nodes, knowledge of their coordinates is sufficient. However, latency data is neither metric nor stable. Scientific research proposed different algorithms to produce and refresh coordinates. All of them can be judged using some of the previously mentioned measures for the quality of a map, accuracy, completeness, or for example the communication overhead of assigning coordinates. In this thesis I analyze these algorithms and propose a novel scheme, Hierarchical Vivaldi, in the main part of this thesis (chapter 6 to 8). As I analyze network coordinates in fully decentral peer-to-peer overlays, I focus on decentral algorithms.

These overlays hardly require an introduction: Peer-to-peer technologies play an important role in modern applications. By locating data at the end-nodes the technology creates benefits, such as bandwidth saving, scalability and resilience. There are plenty of examples for successful P2P services, such as Spotify and of course Skype.

Recently, another technique promising scalability became popular. The cloud computing concept pushes the application logic back into data centers. However, inside these data centers P2P techniques organize the single nodes. Furthermore, in non-profit systems that do not have an appropriate budget available P2P is the only vital option. For example, file sharing based on BitTorrent does not depend on any external storage services and accounted for 34.3 % of North American traffic in 2010.

Goals

Therefore, the primary goal of this thesis is to improve the overlay's awareness of its underlay. Peer-to-peer applications form overlay networks whose topology does not need to reflect the underlying network's topology. If it did, both the overlay and the underlay could benefit. Applications in the overlay would experience a lower communication latency and an increased throughput. The underlying network would have to carry less traffic.

To improve that awareness I analyze the existing network coordinate algorithms. I focus on truly distributed algorithms with no global knowledge requirements. The current de-facto standard is the Vivaldi algorithm, along with an optimization of Vivaldi called Pyxida. I will evaluate both algorithms to determine their performance under ideal circumstances, and I try to deduce a set of recommended parameters.

After deriving a sound understanding about both algorithms, I propose the Hierarchical Vivaldi algorithm, a hierarchical enhancement to Vivaldi. Hierarchical Vivaldi is a practical algorithm that improves over Vivaldi by an order of magnitude. I verify that algorithm's performance in both simulations and real life evaluations.

I have a special focus on the re-usability of the building blocks of my thesis. I start out with the IGOR system, as provided by the doctoral thesis of Kendy Kutzner. Using that basis I extend the components, following Rhea's appeal of broadened interfaces. I provide an ecosystem of P2P components that the community can re-use to develop novel applications far beyond the typical "distributed hash table" scenario. The use of these components for my simulations of Vivaldi and Hierarchical Vivaldi allows me to reaffirm the re-usability of that ecosystem myself.

Overview

The remaining chapters of my thesis are concerned with the following topics:

Chapter 2 introduces the topic of round trip time related research. I start with an overview of the Internet's current structure and components. I identify sources of packet delays and introduce research that focuses on latency optimization.

Chapter 3 describes the P2P paradigm. The need to locate information at edge nodes, led to the development of structured overlays, which allow efficient searching. The chapter briefly introduces algorithms and concepts that have been developed over the last decade.

Chapter 4 introduces the Internet Grid Overlay Routing (IGOR) ecosystem, which is designed to foster the development of P2P applications. Currently, P2P still consist of application specific software solutions. I extended the key based routing (KBR) daemon of Kutzner's thesis [Kut08] into an re-usable ecosystem including a proximity enhanced KBR daemon and a distributed hash table (DHT) library. My work picks up Rhea's appeal of "slightly broadened interfaces" [RGK+05] and provides a high degree of extensibility and customizability.

Chapter 5 discusses the predominant algorithms that use proximity information for overlay optimization. It introduces network coordinates as means of decentral latency prediction. Using network coordinates, each peer can determine mutual round trip times by comparing their virtual coordinates, similar to looking up distances on a city map.

Chapter 6 proposes the Hierarchical Vivaldi algorithm, a multi-dimensional approach of network coordinates for latency prediction. Its hierarchy of embedding spaces provides a new, highly accurate error measure to distinguish nodes with well determined coordinates from others with only vaguely determined coordinates.

Chapter 7 presents an extensive simulation of Vivaldi and the Pyxida algorithm. It studies the optimal parameter combinations for both algorithms. Furthermore, it evaluates Hierarchical Vivaldi using simulations based on realistic traffic sources. Using the previously studied parameter settings I compare the performance of Hierarchical Vivaldi to both Vivaldi and Pyxida.

Chapter 8 confirms the findings from the previous simulations in a planetary scale measurement study. Included in the massively popular BitTorrent client Vuze, I evaluated Hierarchical Vivaldi over a period of five month. During that time period clients sent over 40 billion measurements back to the statistic collecting node. Finally I draw further insight into the performance of network coordinate algorithms using the collected measurement data.

Chapter 9 concludes the thesis and provides an outlook.

Published Work

Parts of this thesis have been published:

Chapter 3:

Benedikt Elser and Thomas Fuhrmann and Georg Groh: *Group Management in P2P Networks.* Proceedings of the 2nd IEEE Workshop on Grid and P2P Systems and Applications (GridPeer 2010), August 2–5, 2010, Zurich, Switzerland

Chapter 4:

Benedikt Elser and Thomas Fuhrmann: *A generic KBR library with built-in Simulation Capabilities.* Proceedings of the 7th International Workshop on Modeling and Simulation of Peer-to-Peer Architectures and Systems (MOSPAS), Istanbul, Turkey, 2011

Chapter 6,7:

Benedikt Elser and Andreas Förschler and Thomas Fuhrmann: *Spring for Vivaldi – Orchestrating Hierarchical Network Coordinates.* Proceedings of the 10th IEEE International Conference on Peer to Peer Computing (P2P), Delft, The Netherlands, 2010

Chapter 7:

Benedikt Elser and Andreas Förschler and Thomas Fuhrmann: *Tuning Vivaldi: Achieving Increased Accuracy and Stability.* Proceedings of the 4th International Workshop on Self-Organizing Systems (IWSOS), Zurich, Switzerland, 2009

Chapter 8:

Benedikt Elser and Thomas Fuhrmann: *Here is Your Peer! – Locating Peers on a Regional Level with Network Coordinates.* Proceedings of the 11th IEEE International Conference on Peer to Peer Computing (P2P), Osaka, Japan, 2011

2. Round Trip Time Research

The major contribution of my thesis is an improved algorithm for latency prediction. The purpose of a prediction is the omission of a measurement step, prior to actual communication. For example, nodes could select the nearest replica out of a number of servers, without measuring latencies to all of them. Latency sums up from a variety of sources and events, such as routers or transport protocols. Additionally it fluctuates due to a variety of factors e. g. route flaps or failovers. Therefore, this chapter introduces the reasons for latency and its various sources, before turning to latency prediction in Chapter 5. Hence, I will start with a brief introduction into the architecture of the Internet's architecture in Section 2.1. In Section 2.2 I will analyze that structure for sources of latencies. Furthermore, I will introduce current work on latency research in Section 2.2.

2.1. The Internet Architecture

The Internet is a computer network that interconnects millions of computing devices throughout the world [KR07] On its highest functional level it consists of hosts or end-systems that are connected to other hosts via a path. On a lower level this path consists of passive communication links, like optical or copper cables and active packet forwarding entities, such as routers switches or bridges. Hosts are identified in a common address space, defined by the Internet Protocol (IP). They exchange messages along a path,which are split into packages that are transmitted independently across the network. That form of delivery implies no initial path allocation like in the plain old telephone service (POTS).

The independent transmission, the so called "stateless" nature of Internet protocols is a direct result of their design principles [SRC84]:

> *[The end-to-end principle] suggests that functions [...] can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system.*

The most prominent example of this design principle is the Internet Protocol Suite's Transmission Control Protocol (TCP). It is layered on top of IP and provides a connection oriented ordered reliable communication channel with error correction, flow and congestion control [CDS74]. As a result of the end-to-end principle TCP's packet transmission is flexible enough to provide connections even when facing changes in routing decisions

or failures. Furthermore, because of the end-to-end principle routers could be built more space efficient and focus on their routing duty.

2.1.1. The Internet Anatomy

Figure 2.1 presents a snapshot of the worldwide submarine cable system in 2008. It is the back end of Internet communication. Satellite transmission faces the problems of a shared medium, while being expensive. Hence, fibre optic cables connect the continents. Clearly North America is the central communication hub. Nearly each European submarine cable connects to the New York region, while only two cables exist to Central and South America. The decentral architecture of the Internet, its fault tolerant design and the redundant provision of cable links allows the network to operate even under dramatic circumstances. During the earthquake in the western Pacific region in 2011 half of the cables were damaged [Jou], yet Japan was still online.

From the latency perspective the figure reveals also information about the expected performance. Latency is bound to the speed of light in fibre optics, which is 200 000 km/s. A signal across 1000 kilometers will take 5 milliseconds to propagate over this setup. Therefore connections from Europe to Australia (approximately 16 000 km) are lower bound at a RTT of roughly 160 ms. However, the actual RTTs are an order of magnitude higher due to various influences, such as routing cf. Section 2.2.

2.1.2. Routing

Distributing route information about every available host to each end-host is obviously not a practical solution. Hence, the Internet is organized in a hierarchy that is predetermined by the IP name space. Each device is at least aware of the *default gateway*, a host, which forwards packets, if the destination is not in the local scope (subnet). Dedicated routers contain tables of prefix / host combinations. A route is selected by matching the target address against each prefix. The longest prefix match identifies the most specific route and determines the packets next hop. This algorithm repeats until the packet reaches the destination host. Each routing decision is based upon local knowledge only, and is often described as *best effort* service. Each routing decision adds a processing delay to the path latency.

Connections between different networks creates the need for buffering at routing devices. Packets need to be stored until the router is able to process them. For example, a packet burst from two different 100 Mbit networks, targeting a third 100 Mbit network, will consume all available bandwidth. Hence the router queues packets if that queue is not saturated (congestion). Otherwise the router has to drop packets.

The TCP protocol uses dropped packets as input to a congestion avoidance algorithm that throttles sending of packets (implicit congestion notification). It relies on timely notification of these events. TCP's success proves the effectiveness of this mechanism, however, there are corner cases of "chaotic" behaviour [VB00]. Veres et al. claim that

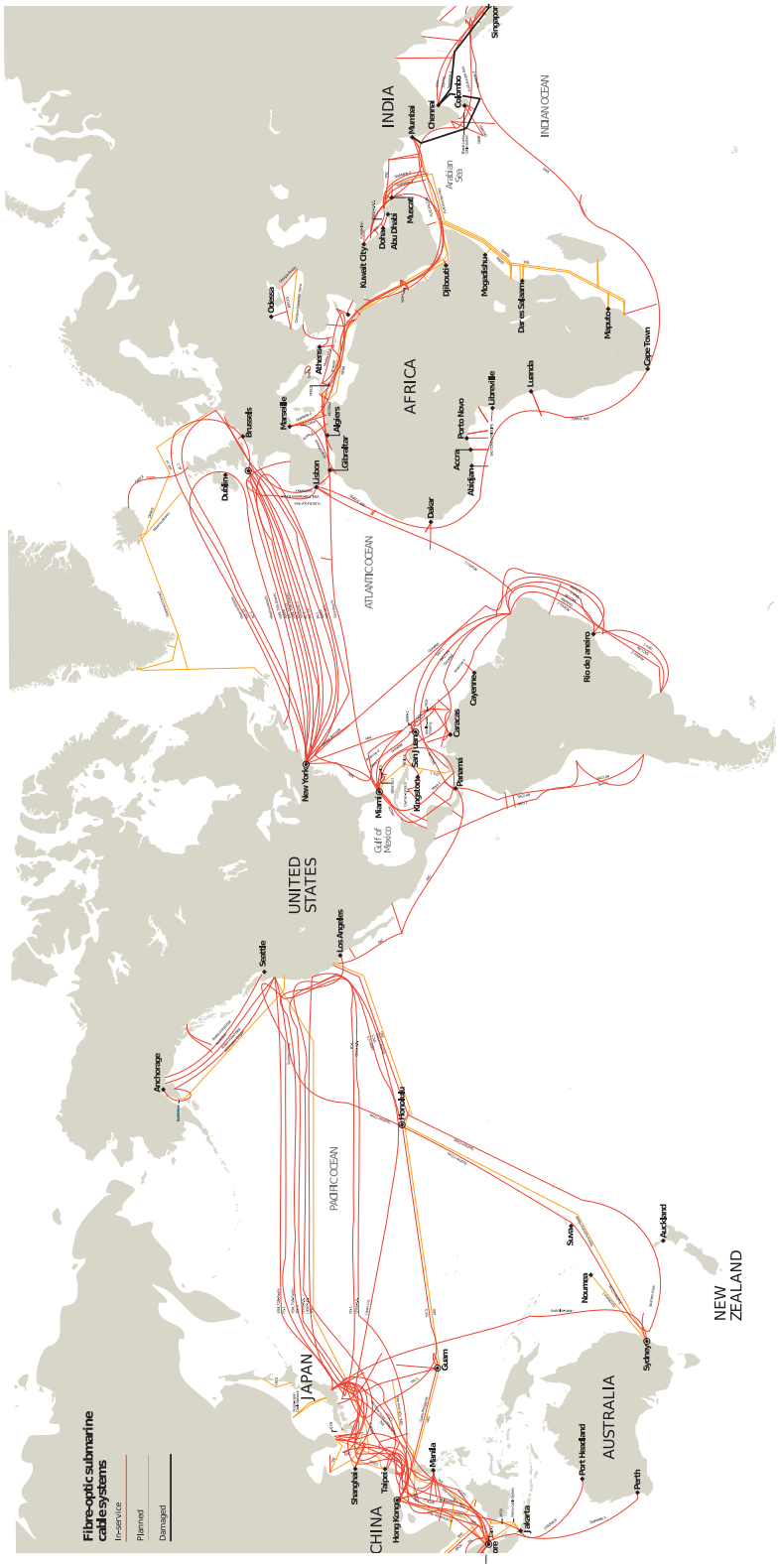

Figure 2.1.: The worldwide submarine cable system in 2008 [Tel].

TCP's implicit congestion avoidance contributes to the fractal nature of Internet traffic in contrast to being a pure stochastic process. To avoid saturating a queue, which results in packet drops, proposals for explicit notification through active queue management exist, such as Random early detection (RED) [FJ93].

The impact of multiple queuing along a path is a major contributor of latency as discussed in a recent study [KMS+09]:

> *Beyond propagation delay, high RTTs could also be the result of packets getting queued up somewhere along the path between the clients and the nodes serving them.*

I will discuss the problem of "bloated buffers" in Section 2.2.3.

2.1.3. Autonomous Systems

The Internet's structure is tiered, following a hierarchical organisation. Independent operators e. g. ISPs, universities, etc. control their (unique) routing prefix. They are commonly referred as autonomous systems (AS). Contracts manage the delivery of packets that terminate in other ASes (inter-domain traffic). Either an AS pays another upstream AS to deliver inter-domain traffic, or two operators route non-transit traffic between their networks, without charging one another for traffic (peering).

ISPs follow different incentives: First, they strive to attract a maximum number of customers e. g. by bandwidth or price vantages. Second, they want to minimize their costs for traffic they need to deliver outside their network. Third, they are reluctant to process packets that transit their network. The third point leads to the use of hot potato routing in peering situations [ZLPG05], a technique that delivers incomming packets to the closest egress point, to minimize processing time in contrast to latency minimization Section 2.2.2.

Depending on their size and peering requirements ISPs are divided into three Tier categories, where Tier 1 ISPs do not need to purchase any inter-domain traffic, Tier 2 ISPs pay for some traffic and Tier 3 ISPs need pay for any exterior traffic. Depending on their connectivity ISPs are often reduced to belong to the well connected network ”core” and the ”edge”. This is illustrated in Figure 2.2. Few Tier 1 ISPs are located in the center part of the circle, colored in orange, while the majority is at the circles boundaries, colored blue with occasional peerings.

2.1.4. BGP

The Border Gateway Protocol (BGP) is used in the inter- and sometimes intra AS routing decisions. For simplicity reasons I focus on the inter-as roting (eBGP), iBGP is used only in large ISP networks. BGP is a fully distributed path vector protocol. It incorporates network policies into its path decision. Commonly these consist of combinations of peering and cost information. Based on these policies routers dynamically updates their forwarding tables and reflect changes in the network state instantly.

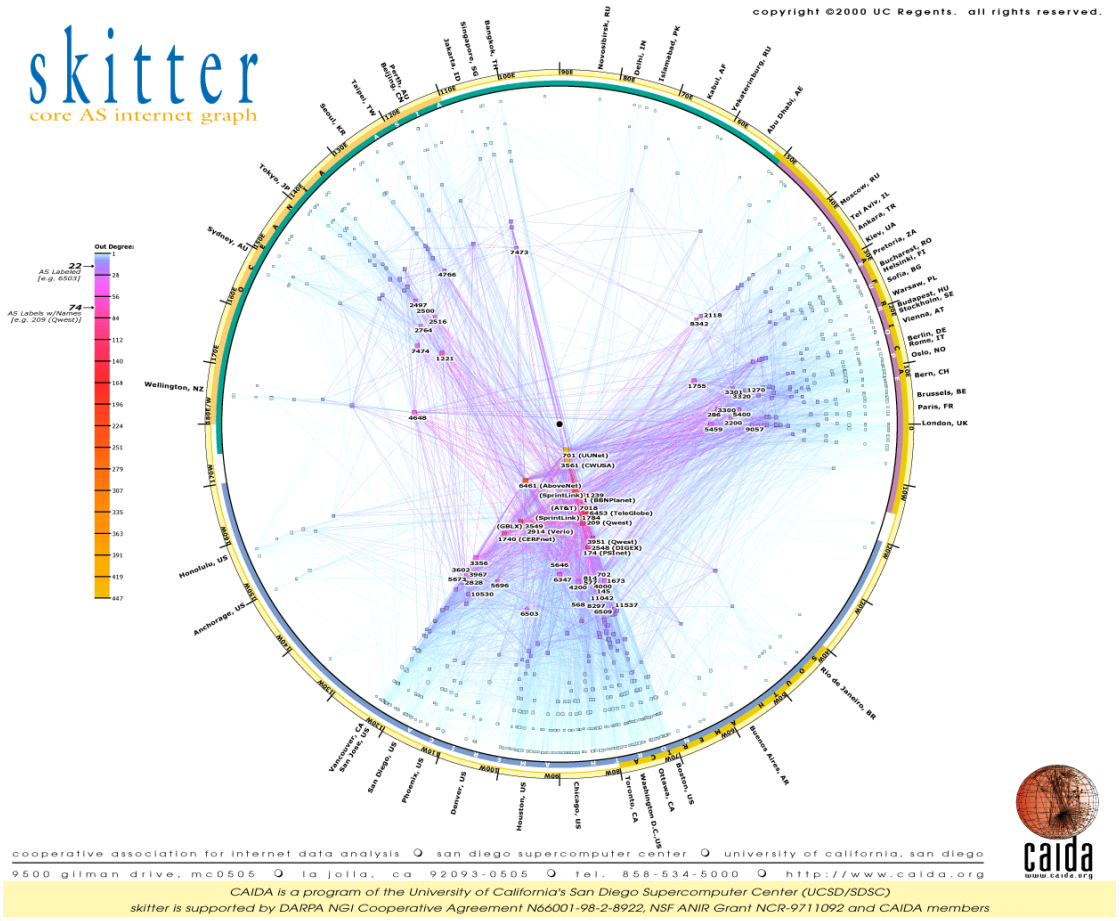

Figure 2.2.: AS connectivity illustrated by the cadia project [DKR05].

Local BGP misconfigurations can have drastic impact on the global routing behaviour [MWA02; CKV10]. For example, a national BGP blockade of the popular video streaming site *youtube.com* in Pakistan, turned into a global one, due to a simple misconfiguration. Hence all traffic would be routed via Pakistan [Ré08].

2.2. Analysis of Round Trip Times

The previous section introduced the basic architecture of the Internet. Along with it, I presented the example of a connection between Europe and Australia, which is bound at 160ms. However, latency measurements between end-host exhibit delays three times higher (cf. Figure 2.3) or even more [SSW10b].

That example latency of 160ms is a theoretical limit, it is the *transmission delay*. Fibre optical systems need processing to amplify their signal after a distance of approximately 110 km, which additionally increases latency. Furthermore, each routing decision introduces additional delays, as it requires excessive processing of packets: Packets need to be decoded from the medium, processed an encoded back onto the medium (*processing delay*). Furthermore if a packet can not be processed immediately, it has to be queued (*queuing delay*)

Another major contributor to delays is, the actual routing path. The Internet is a network of autonomous systems and little is known about the actual inner workings of an AS. It is however certain, that ISPs have no incentive to prioritize traffic passing through. As Leighton puts it:

> *The path between sender and receiver is a "heterogeneous infrastructure that is owned by many competing entities and typically spans hundreds or thousands of miles" [Lei08]*

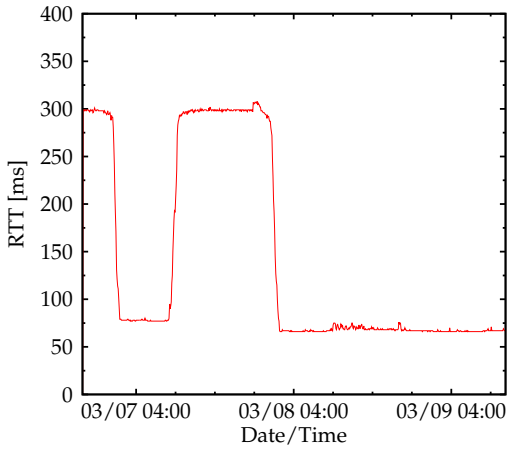

Figure 2.3.: RTT during a continuous measurement between *planetlab4.lublin.rd.tp.pl* and *planetlab-1.iscte.pt*.

In his awarded paper "End-to-End Routing Behavior in the Internet", Vern Paxson concluded that round trip time behaviour is imminent unpredictable [Pax97], hence research focuses to find the reason for this misbehavior, which will receive attention in this chapter. Figure 2.3 serves as introduction of the problem we are facing. RTT does not only vary in small scale, but also shows fluctuations at a magnitude of 3 times.

Starting at approximately 300 ms the RTT drops to 70 ms for a number of hours, before returning back to 300 ms. A second drop after half a day seems to stabilize the RTT at 80 ms. The respective data was captured during a measurement between PlanetLab hosts conducted in 2007 cf. Chapter 7. The illustrated data demonstrates the effect and the presence of routing changes during standard network operation.

2.2.1. Routing Paths and the Shortest Path

A routing path, that is based on the geographic shortest route, should optimize the transmission delay. Various parameters influence that path, however, in [MUF+10] Mühlbauer et al. conclude that despite substantial research and age, the impact of these parameters on the resulting route is still unknown. Therefore they investigate the impact of parameters, such as the size of the autonomous system, peerings and policies etc. to route stretch. In their study they are able to reassure the findings of Krishnan et al. [KMS+09], concluding:

> *Overall the majority of routes incur reasonable stretch. [MUF+10]*

However, they were unable to optimize routing paths to reflect geographical distances. Their proposal is to enrich the intra domain protocols with geographic information, if *end-to-end path quality is to become more important in the future* [MUF+10].

In [SSW10a] Schwartz et al. analyze the stability and diversity of end-to-end routes, using a longitudinal analysis of data from 2006 and 2009, in [SSW10b] they analyze the origins of delay variations from the same data. They analyze whether latency variations differs significantly due to different routes. In both studies they confirm the exceptional position of academic networks. These networks are usually neither load balanced nor traffic shaped. Therefore they exhibit higher route stability and lower latency variance, than commercial networks. Furthermore their second study confirms, that delays are mainly introduced by *"changes along the route itself"* [SSW10b].

2.2.2. The Impact of Peering

Both studies [WMW+06; ZLPG05], investigate the role of routing and routing events in detail. Wang et al. [WMW+06] found that in spite of the availability of redundant paths, routing still suffers from missing redundancy information. Furthermore BGP's built-in fault tolerance comes at the cost of loss bursts that are especially high on failover events. These bursts manifest in huge latency increases over periods of hours.

Zheng et al. [ZLPG05] discuss the use of hot potato routing in peering situations. As discussed earlier in this thesis, the economic incentive of an ISP is to minimize the costs of inter-domain traffic. Hence, they try to deliver that traffic to the closest egress point as fast as possible. That behavior is not only malicious in terms of latency: The authors found this to be responsible for triangle inequality violations (TIVs cf. Section 2.3.5) in both an asymmetric as in a symmetric routing relationship.

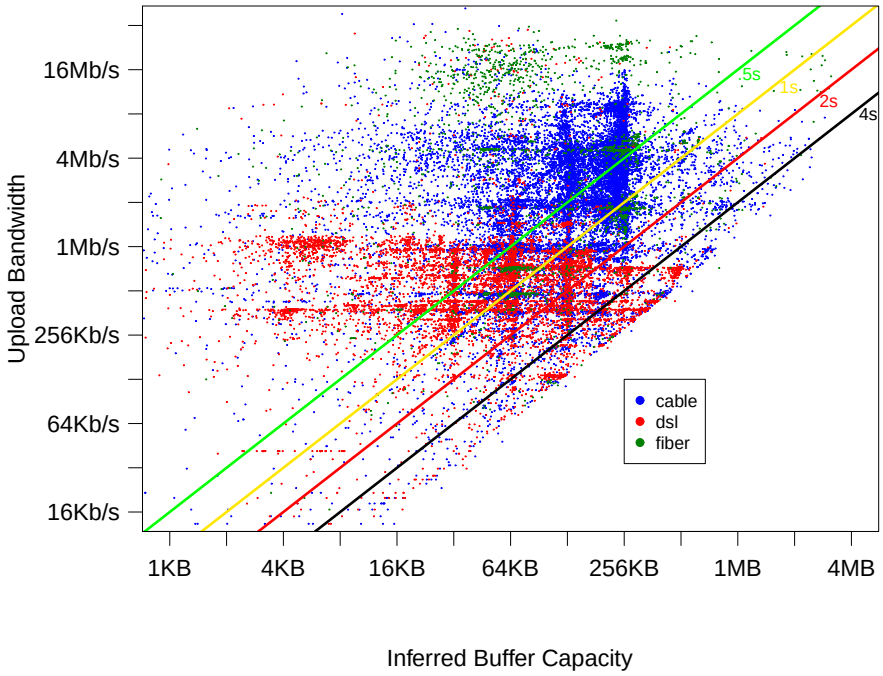

Figure 2.4.: Uplink buffer size, inferred by ICSI Netalyzr [KWNP10].

2.2.3. Bufferbloat

Equipment of residential broadband Internet access has always received much attention cf. [DHGS07]. A topic that has recently caught attention in network latency research is the negative impact of overdimensioned buffers inside network hardware on overall network performance [Get11; GN11]. So far, the bandwidth delay product has been considered the "gold standard" concerning buffer sizing. However, in a broad study conducted by the International Computer Science Institute (ICSI), their tool for testing end user connectivity – the Netalyzr [KWNP10] – found a sizable fraction of network hardware to be significantly over-buffered.

Figure 2.4 shows Netalyzr's inferred buffer size plotted relative to the upload bandwidth of the residential Internet access. Clearly most cable network hardware is equipped at least with 32KB, however up to 256KB are common. The lines in the plot indicate how long it takes to empty a buffer of the respective size. For example, a 2Mb/s DSL modem will fill a buffer of 256KB in one second (second line from left). In contrast, the bandwidth delay product at a RTT of 50 ms (typical for DSL links cf. [HUKC+11]) recommends a buffer size of only 100KB. Obviously, the time to fill the buffer is independent of the actual latency.

Until this buffer is filled, the TCP congestion control algorithm is unable to detect the bottleneck bandwidth of the connection. Rather, TCP slow start will exponentially increase the size of the send window. Thus, packets will be sent at a significantly higher rate than the actual bottleneck bandwidth. One effect is that a single TCP connection

may be able to slow down all other connections crossing the same bottleneck, as it cannot receive the required throttling notification in time. Another effect is that latencies drastically increase, as many packets start getting queued and thus will have to wait *buffer size at bottleneck link / bottleneck link bandwidth* seconds for their transmission – a time span that obviously grows linearly with larger buffer sizes.

2.2.4. Random early detection

Section 2.1.2 mentioned the random early detection algorithm as means of active queue management (AQM). AQM monitors the queue size inside a router and explicitly marks packets with a congestion header. The marking process happens with a probability relative to the queue length. Hence, the receiver of a marked packet can inform the sender, that contributed to the congestion, to throttle sending. Explicit Congestion Notification (ECN) defines fields in both the TCP and IP protocols header for active queue management. There are several extensions and competing proposals including RRED [ZYCC10], an extra robust RED algorithm and RSFB [ZYC09], a stochastic algorithm.

AQM should have eliminated the bufferbloat problem years ago, and ICSI's observations should be uncommon or impossible. Nevertheless, AQM methods appear to be turned off by default in a large fraction of deployed routers. Older hardware is often incompatible with the ECN bits. Network operators even tend to clear ECN bits on the way from the sender to the receiver, in other cases they get ignored or misinterpreted [BBB11]. Furthermore, the RED algorithm is known to be flawed in at least two different ways. Recently et al. proposed the CoDel algorithm [NJ12], designed to solve REDs design flaws.

2.3. Current latency research

After introducing the structure of the Internet and the sources how latency sums up, this section presents current research on how to mitigate or even use delays. Initially I motivate latency optimizations, then I present current research and conclude with a examination, why latency optimizations can not be left to the ISP.

2.3.1. Why optimize for latency?

In 1996 Cheshire concluded in [Che96] that typically bandwidth is not a problem. Commonly bandwidth problems can be mitigated in a flexible way. For example, residential bandwidth can be extended by purchasing more bandwidth. The Akamai report [Bel10] sees this happen. However, latency problems **last**. The same home network that can be equipped with a second subscriber line, cannot buy lower latencies beyond physical limits. Remember that we already concluded in Section 2.1.1 that Australia is bound to an absolute minimum delay of of 80ms.

> *Regardless of the equipment or protocols you use, your data cannot exceed that theoretical limit. This limit equals the delay between when a packet is sent, and when it is received, aka latency [Dou].*

Furthermore, latency is the more useful information. Latency measurements will not fluctuate that enormous facing path changes. Bandwidth is much more dependent on the exact path (cf. [FJJ+01]).

Hence, the goal of this thesis is to provide stable network coordinates. If a peer is presented with different replica, it should be able to choose the latency optimal choice instantly.

2.3.2. Bandwidth optimizations

Although this thesis focuses on latency, I will briefly introduce the Thunder-Dome series of algorithms [DMMP09; DMMP10]. The Thunder-Dome algorithm schedules bandwidth measurements among hosts in a decentral fashion. Previously, algorithms relied on *a priori* knowledge of the upload bandwidth. The key problem Thunder-Dome addresses is the detection of that bandwidth. A misidentification arises easily, as the computed bandwidth between two random hosts is the minimum of a sender's upload capacity and the receivers download capacity: $min(upload_{sender}, download_{receiver})$. Hence, the system schedules a number of *pair-wise* "tournaments" to determine the upload bandwidth in a distributed and bandwidth conserving fashion. The winner of such a "tournament" is reexamined until it determines a node, which is well provisioned enough to correctly detect a node's upload limits.

2.3.3. Anycast

One dilemma of a content provider is its dependence on the path between him and his customers. Typically a path crossing numerous ASes with peerings, uplink contracts and routing anomalies lies between them. This is a major obstacle for companies that depend on low latencies. For example, Google tries to replace the traditional desktop with pure web based solutions. They discovered, that an increase of 0.5 in latency is not tolerated by Google's users [ES09]. However an ISPs has no such incentives to optimize is network for low latencies. Quite the contrary, the rising traffic demands of e. g. video streaming [San10] mean higher costs.

Ideally content would be available in the operators own network, which increases available bandwidth by avoiding upstream bottlenecks. This is the design idea of *Content delivery network* (CDN). CDNs replicate cachable content close to the end user, and deliver content transparently from that latency closest cache (edge node, ghost) instead of the "original" source. For example, in Google's CDN 75% of prefixes have a ghost within 1000 miles, which translates roughly to 20 ms at the speed of wire [KMS+09].

Technically, there are two common ways to accomplish locality. Typically CDN providers e. g. Akamai will reply to DNS queries with the replication server that is

latency closest to that peer. These "ghosts" have different IP address, but the same hostname. The second method, is redirecting clients via `HTTP 300` headers.

CDNs use various informations such as provider support, geographic location and network measurements to determine which replica a client should use. This topic is still a field of optimization. Recently Google proposed a change to the recursive resolution of DNS lookups. In the recursive setup DNS servers forward queries to the authoritative nameserver instead of letting the client forward it. However, in that case a CDN cannot come up with the best replica, because the query lacks information about the original requester. Hence, Google submitted an IETF draft to include the requesting client IP addresses in the DNS query header [CvdGLR10]. Due to privacy concerns it was later revisited to include only client subnet information [CvdGLR11]. Although the proposal did not leave status of a draft, Google and partners already implemented that technology in production systems [Inca].

However, ”close” placing does not always yield the desired effect, as a study conducted inside the CDN of Google by Krishnan et al. found:

> *Our analysis [...] of thousands of prefixes across the Internet shows that more than 20% of paths have an inflation greater than 50ms, even though most paths are between clients and nearby nodes. [KMS+09].*

The same study found latency problems in regions, whereas other peers from the same region experienced latencies ten times lower. The investigation of these phenomenons reaffirmed the two sources of latencies. First, the path might be flawed by routing circuits, which is a serious misconfiguration. Second, a router on the path might be congested, impacting client significantly.

2.3.4. Internet Tomography

Discovering a network’s anatomy, e. g. delays, loss rates or topography information, has always received attention. Infrastructure providers are reluctant to the publication of information about their networks. A reason for this is the notion of companies, to keep their information as business secret. Furthermore on high loaded links, measurement traffic, competing with business traffic is unwelcome. Investments into a monitoring infrastructure for public consumption are uncommon. The only public available information are BGP routes, which define the "interfaces" an ISP offers to the outer world. Tools, such as cyclops [ZLMZ05], or the pinger project [MC00] breathes life into that ”principal structure” of the network.

Hence, the scientific community proposed ”Internet Tomography” [Pax97; CHNY02; CCL+04] as methods of investigation, these black-boxes. Internet Tomography focussed on the discovery of network parameters by a huge number of sophisticated tests. Today the most visible tool that was developed during this ongoing effort is `traceroute`.

However, there are various other tools e. g. `clink` that precisely detects a connections bandwidth. Internet Tomography follows two different objectives.

The first is link level parameter estimation, based on path measurements. By sending packets to different receivers, assumptions about their common intermediate links are possible.

The second is path level traffic quantisation, based on link level measurements. By measuring traffic on an intermediate link, the direction of packets can be inferred.

Both approaches lead to inherent random data. Therefore, statistical techniques are used to analyze the result.

Internet Tomography created the tools for projects like rocketfuel [SMW02], which tries to derive data about the anatomy of ISP networks from measurements. Furthermore it is the underlying technique for e. g. the bandwidth estimation component of ICSI's Netalyzr [KWNP10].

2.3.5. Detour Routing

"Detour Routing" sums up algorithms that change the Internet's standard routing behavior. Instead of a best effort service, specific optimizations can be applied. For example Akamai's SureRoute algorithm [Tec] uses detour routing initiated from Akamai Edge nodes, to redirect users via the fastest path to an origin of *non cachable* content. That service is offered both for performance reasons, and for failover situations, where routing failures inhibit the direct path from customer to provider.

The PeerWise [LBL+09] algorithm, implements that service using a P2P network. It is built upon existing Detour routing frameworks, such as Detour [SAA+99] or One Hop [GMG+04]. However PeerWise exploits triangle inequality violations and detects them via the Vivaldi network coordinate algorithm. I will first introduce triangle inequality violations, before describing PeerWise:

Violation of the triangle inequality (*Triangle Inequality Violation (TIV)*) are a commonly witnessed phenomenon on the Internet. Commonly direct path between two nodes is received as the "shortest path". This reflects geometrical fundamentals, where in a triangle of nodes *A,B,C*, the shortest path between two nodes has to be the direct connection. The length of the path between node *A* and *B* will always be shorter than the path *A - C - B*. However, the Internet does not reflect a metric space. Therefore a drastic fluctuation in round trip times *may* yield such behaviour and detours via another hop may provide smaller latencies.

In [WMW+06] Wang et al. study the causes of TIVs. They identified hot potato routing, along with private peerings and interior level routing probles as a major sources of TIVs. From their analysis, they conclude that TIVs are not a structural anomaly, but a direct outcome of the Internet's design and thus persistent. Measurements from a study by Lumezanu et al. [LBSB09] confirmed these phenomena and also found them to be quite common, varying with time. Their study found routing changes and queuing delays

as major sources of TIVs. Furthermore, they narrowed the lifetime of a TIV down to lie between 5 hours and one day.

In their PeerWise system [LBL+09] system, the authors successfully exploited benefits of TIVs. Each violation represents a potential shorter path between two nodes, involving a detour via a third node. However, the faster path needs also more traffic for that node. Therefore the algorithm is a broker among nodes, that ensures benefits for all three parties, by searching for detours that benefit that third node.

> *The key idea is that two nodes can cooperate to obtain faster end-to-end paths without either being compelled to offer more service than they receive [LBL+09]*

The authors were able to prove that for 50% of nodes in their example data sets detours existed. The mutual advantage requirement shrinks that fraction for some nodes completely, others still exists. PeerWise employs network coordinates to predict and select detours, as these algorithms are unable to cope with TIVs and thus exhibit a high degree of error. This will receive broader attention in Section 5.5.

2.3.6. Prediction vs ISP knowledge

The first idea that comes to mind, when trying to predict latencies is support from an ISP. The operator of a network is the presumably most competent partner to tackle this issue. Solutions, how this can be accomplished are described in Section 5.12, this section rather describes the relation of customers and ISPs.

One main application of latency prediction are overlay networks, such as file sharing applications. Clearly ISPs' reaction towards the P2P traffic generated from file sharing was not welcoming. Both ISPs and users started a competition towards creating and circumventing barriers for P2P traffic. Firstly ISPs attempted to limit P2P traffic by using traffic shaping and blocking, based on targeted ports. However, applications switched to use random port numbers. To identify P2P traffic with random port numbers ISPs invested in Deep packet inspection. That move resulted in even bigger buffers, to allow scanning of data, which lead to higher latencies. Users responded by encrypting data. ISPs also attempted to redirect users to hidden caches in their network. This practice is legally questionable, since this could be interpreted as a distribution of potentially copyrighted data.

Therefore users will be reluctant to solutions offered by their ISP. Too big might be the incentives for ISPs to keep traffic local, even if lower latency peers might exist outside of their network. A recent study initiated by Google and conducted by Mueller et al. [MA11] sheds light on the degree of ISP's interference targeting BitTorrent traffic. Using the "Glasnost" software setup [DMG+10] in Google's Measurement Lab [1], users could

[1] `http://http://www.measurementlab.net/`

test their Internet connections for signs of throttling of BitTorrent traffic. Starting in 2008 these measurements show indicators for throttling for most ISPs. The intervention of the Federal Communications Commission (FCC) against the US provider Comcast provoked a strong reaction and resulted in less throttling. Throttling still exists, for example, the German provider "Kabel Deutschland" was convicted of throttling in 44.5% of 450 tests [2]. The specific result for that provider is interesting, as our measurements in Section 8.2.2 led to the same conclusion while investigating Vivaldi anomalies.

[2] http://dpi.ischool.syr.edu/countries.html

3. The Peer-To-Peer Paradigm

The P2P paradigm has become popular about a decade ago. The rediscovered focus on end hosts symbolizes a return from the client/server architectures of the 90s back to a decentral structure.

In its very beginning, the ARPANET, as the Internet was called at this time, consisted of computers, each equal among them. It was initially a network between the Universities of California (Los Angeles and Santa Barbara), Utah and the Stanford Research Institute. The network was built, *to share computing resources around the USA* [TH09].

After the Internet became a mature technology, access was granted to less powerful nodes, such as consumer level devices. In fact the overwhelming penetration of "the web" in modern society produced an enormous number of end users. That imbalance between service providers and consumers naturally led to the predominant client server paradigm discussed in the next section.

However, in the last 10 years, with the rising demand for Internet access and services, two developments have led to the renascence of the P2P paradigm. First, servers were unable to cope with the rising consumer demand e. g. for video streaming. Second, residential bandwidth reached rates, that allowed home users to meet the demands of contemporary services, such as video streaming. Because end users share resources, the P2P concept reaches "back to the roots" of the Internet, although the term P2P has only been in use since 1999 along with the Napster system.

P2P research proposed the algorithms that will receive attention in this chapter. They can be split into different flavors: Unstructured and structured overlays, with occasional proposals of hybrid solutions. Unstructured overlays cause little maintenance overhead (cf. Section 3.2). However, they need to *search* for information, like the needle in the haystack. Structured overlays efficiently *locate* individual data items (cf. Section 3.3). Their strict organisation guarantees success, provided the item is available somewhere in the overlay. If combined with proximity-awareness techniques as described in cf. Chapter 5, they can achieve a good routing efficiency. As a downside, maintaining the required structural constraints of the overlay requires a potentially high overhead.

Hybrid P2P systems (cf. Section 3.4) migrate some tasks, such as billing, to a central component, while retaining load intensive tasks distributed. One of the most prominent examples for hybrid systems is BitTorrent, which I will introduce along with its well known *Vuze* client in Section 3.4. That very client was used to conduct experiments "in the wild" in Chapter 8.

These different P2P structures will receive attention in the course of this chapter. I will introduce the most outstanding examples of each technique alongside the theoretical reflections.

3.1. The Client/Server paradigm

The predominant architecture in computer networks is the *Client/Server* paradigm. When a client requires a service, it connects to a service provider, a so called server. That server fulfills the clients request. This model has a lot of benefits, for example:

- The Client could be very simple, in complexity of both hardware and software.
- The Server is holding software logic and data in one central place, which ensures consistency of data and avoids fragmentation.

In recent years the demand for bandwidth intensive services increased [Cis10]. For example, in 2010 online video consumed 36 % of the Internet traffic. Additionally, the number of clients has increased, because of the overwhelming penetration of "the web" in modern society. Therefore, servers face scalability issues in terms of bandwidth. One method is to provide even more servers, placed closer to the users (cf. Section 2.3.3). However, there is another approach. At the same time as servers see higher demands, clients get equipped with increasing resources. Nielsen's law [Nie98] predicts an annual 50% increase in user's connection bandwidth. Today Clients' bandwidths are sufficient for providing a service, such as video streaming, which currently has a bit rate of 512kb/s in H.264/AVC MP [OBL+04]. The idea to directly connect clients defines a new paradigm. Steinmetz et al. [SW05] define a P2P system as,

> *a self organizing system of equal, autonomous entities (peers) [which] aims for the shared usage of distributed resources in a networked environment avoiding central services.*

In the words of the Client/Server paradigm, every client acts as server. Ideally, every client uses and contributes resources to the network (*decentralized resource usage*). To be able to do so it must operate independently: Each client makes decisions on the basis of local knowledge only (*decentralized self organization*) [SW05, p. 11]. The latter principle expresses a core design philosophy of P2P algorithms. Requiring more than local information leads to abundant communication. That would make the system overly complex and slow. Therefore local design is a fundamental principle.

These P2P networks are also referred as overlay networks cf. Steinmetz et al. :

> *Let $V = 1, 2, ...n$ a set of all n nodes, or peers. Every Peer can be associated with a graph $G = (V, E)$. E is the set of edges $e = (i, j)$ where j is a neighbor of i, i.e.,*

> *there is at least one entry in the routing table of i that uses j as the next node. For edge $e = (i, j)$, i is the source node and j is the target node. The number of edges is denoted by m. G is sometimes called the overlay network of a Peer-To-Peer system. The edges might be weighted, e.g., with the number of entries that use j as the next node or the cost for the traverse of this edge. All edges are directed. [SW05]*

The above definition intentionally leaves the type of node $i's$ identifier undefined. Overlay networks commonly identify nodes by using random IDs, not IP addresses. Thus, they emphasize their independence from existing network topologies.

To reach a target node T in a P2P network, that is not in the neighborhood N_A of a node A, at least one node $B \in N_A$ has to forward that message. The actual routing algorithm is specific to the overlay algorithm and will be discussed in the following sections. The sequence of nodes that forward a given message constitutes the path that the message takes through the overlay. Each step along that path is called a hop.

P2P's robustness roots in its design. *Churn*, the spontaneous joining and leaving of nodes is at the overlay's design very core. It diligently reflects the fact that end users can and will disconnect their machines at any time. In contrast server systems are dedicated machines and will only disconnect in case of failure. End user machines serve many different tasks and thus do not operate for the sole purpose of *serving*. However, compared to the sheer number of servers, they exist in abundant numbers. Therefore, even if one client disconnects, another is there to take its place.

To sum it up, P2P's main benefits are it resilience to faults, abundance of resources, the ease of deployment, self organization and the high degree of decentralization [RD10]. However, not each problem can be easily addressed with simple dedicated algorithms. There are scenarios where a centralized algorithm solves problems more easily. These problems include membership control, privacy issues and the management of distributed applications, which are still under active research.

3.2. Unstructured Networks

When the Napster system [VLO09, p.229] was shut down in February 2001 the fragility of a network with a single point of failure became apparent. The system relied on an index server, which was obligatory to keep the network operational. However, the enormous scalability of the P2P paradigm that was achieved by the equal distribution of workload among clients, sparked interest in the research community. A focus to improve the robustness of totally decentralized networks evolved in the research community. These networks should function in the absence of any central component. No structural constraints are applied to the connections of a node. Only few connections to a tiny subset of all available nodes are necessary.

In spite of the tiny number of connections the so-called small world phenomenon [Mil67] leads to a network where each peer can reach every other peer along a short path. The phenomenon became popular in the 1960s when a sociological experiment found that six intermediates suffice to pass a letter from a randomly chosen person to a specific target person in the US. Hence, also the term "Six degrees of Separation". The experiment demonstrated that although a single person has only a few friends, each of those is friend to a number of others. Clearly, each intermediate in the experiment had to choose carefully that exact friend from her list of acquaintances that would be closer to the messages target, in terms of geographical closeness, social situation etc. . Applied to a P2P network, this means that each node can reach any other node by passing messages among its neighbors. The way nodes are identified and how a message finds its target are details to the specific algorithm. Rodrigues defines unstructured networks as:

> *In an unstructured P2P system, there are no constraints on the links between different nodes, and therefore the overlay graph does not have any particular structure. In a typical unstructured P2P system, a newly joining node forms its initial links by repeatedly performing a random walk through the overlay starting at the bootstrap node and requesting a link to the node where the walk terminates. [RD10]*

Due to the inherent randomness of these overlays, their construction requires elaborate algorithms for locating data. The next section introduces the simplest overlay: The Gnutella [Rip01] protocol. More complex examples include Freenet [CHM+02], which builds a structure on top of a unstructured network, while laying a strong focus on privacy. Furthermore, the BubbleStorm [TKLB07] algorithm also tries to efficiently locate data. Both approaches still cannot reach the efficiency of a DHT (cf. Section 3.3.2). I chose to introduce Gnutella in the following chapter, as it is simple and widely studied in the research community. This will enable the reader to quickly understand a unstructured system without bothering with search or privacy optimizations.

Gnutella

The first true P2P protocol, working in a totally decentralized fashion is the *Gnutella* protocol [Rip01] in its initial (0.4) version. In Gnutella all clients (servents) are equal among each other. They form an overlay network based on their associated `servent_id`. Every peer is connected to a number - commonly up to 10 - of other peers in a mesh topology. These other peers are called the "neighbors" of a node. The number of neighbors is also referred as the degree of a node. The Gnutella overlay is totally randomly connected. Stability facing node churn is ensured by learning new nodes via flooded `Ping` messages.

Gnutella is unable to determine, which neighbor is closer to a target because of the random nature of its neighbors and IDs. Therefore, routing in Gnutella is, in contrast to the small world experiment accomplished using an algorithm derived from the flooding

protocol. A node that searches information broadcasts a request to its neighbors, which in turn will contact their neighbors. The request is flooded in the network equipped with a "time to live"(TTL) that is decreased when passing the search further. When the lifetime counter is zero, the search terminates. This maximal network coverage is called the horizon. The concept is very similar to the time to live (`TTL`) field of the `IP` header. Clients answer to the request using the reverse routing path back to the query's origin.

During Milgram's letter passing experiment intermediate hops where able to choose an optimal next hop. Although the Gnutella protocol is unable to determine that optimum, it will nevertheless find the most efficient routing path. Real letters could not be copied and passed to more acquaintances at ease. Therefore, the sociological study relied on the best effort choice of the next hop. In contrast Gnutella sends the message to every next hop and will naturally include the best next hop. Therefore, the duplicate that arrived first, traveled the optimal path.

In Figure 3.1 two paths between node A and B exist. Each path involves intermediate hops, the solid line needs four hops to reach B, the dashed line two. Furthermore, assume each hop has a latency of one. If A searches for B, both paths will be flooded with a search request, B will receive two messages, but the dotted path will reach B earlier. The example illustrates the downside of flooding: Although flooding finds the fastest path, it congests the network with duplicate searches.

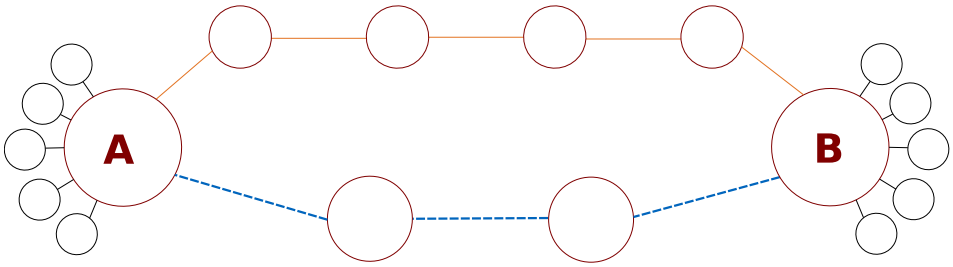

Figure 3.1.: An example Gnutella network. Node A reaches Node B via multiple paths. If every edge has weight 1, queries from A to B reach B first via the dotted path.

The Gnutella network evolved in its 0.6 protocol version due to poor performance. It abandoned the balanced topology, where each node is equal. The system introduced *hubs*, nodes with higher degrees than others that form a core overlay network. Normal nodes attach to hubs as *leafs*. Hubs act as an index server for leaves, respond to their queries or forward them to other hubs.

Unstructured networks are simple in their design and implementation. They do not need elaborate algorithms to keep up a structure of any kind, which is an advantage in terms of message overhead. Therefore, they received a lot of attention from the research community, for testing and improving their behavior [AFS07]. The drawback in terms of message overhead is the need to search by flooding queries across the network. Furthermore, there is no guarantee that a search for content will yield an answer, as there is no structure that helps locating content.

A next generation unstructured network is the recently proposed Equator service [MCM+11] that uses the Barabási-Albert model to avoid flooding. Although it is specialized in locating equivalent servants, its potential for locating arbitrary information deserve an analysis once all details of the algorithm are released.

If the simplicity and the absence of a protocol enforcing the structure of the network makes these networks a better choice than the structured networks discussed in the next section is still open research.

3.3. Structured Networks

The previous section discussed unstructured networks along with their shortcomings. In this chapter, multiple proposed solutions to the problem of locating information will receive attention. Their common denominator is, that, efficient lookups of data items are possible, provided their *key* is known. Structured networks guarantee success, provided the item is available somewhere in the overlay. As a downside, maintaining the required structural constrains of the overlay requires a potentially high overhead.

In [RD10] structured networks are defined as follows:

> *Each node has a unique identifier in a large numeric key space, for example, the set of 160-bit integers. Identifiers are chosen in a way that makes them uniformly distributed in that space. The overlay graph has a specific structure; a node's identifier determines its position within that structure and constrains its set of overlay links.*
>
> *Keys are also used when assigning responsibilities to nodes. The key space is divided among the participating nodes, such that each key is mapped to exactly one of the current overlay nodes via a simple function. For instance, a key may be mapped to the node whose identifier is the key's closest counterclockwise successor in the key space. In this technique the key space is considered to be circular (that is, the id zero succeeds the highest id value) to account for the fact that there may exist keys greater than all node identifiers.*

Commonly these system implement a variant of binary search on their induced data structure. Therefore, routing lookups resolve in $log(n)$. Before discussing Section 3.3.1 and Section 3.3.2 will introduce a separation of layers between routing and higher level functionality in a structured overlay.

3.3.1. Key based routing

Common to early proposed structured networks, was the focus on their data storing capabilities in a distributed hash table (DHT cf. next section). As Dabek et al. pointed out in [DZD+03], a DHT is not the right level of abstraction for generic P2P applications.

This focus hampers the use of structured networks beyond primitive data storage. Dabek et al. proposed the key based routing interface as a common API for structured P2P systems.

The basic KBR API primitive is `route(key, value)`. It is based on an application-dependent `key`, which is hashed into a virtual address space, common to all applications using the same KBR system. Every node handles the received `value` in an application-dependent manner.

That simplified API provides functionality for various fully decentralized applications. The following applications were developed at the Chair for Network Architectures of the Technische Universität München, some as part of the development of this thesis: A chat service [Hud08], a video-on-demand system [KCF05; Nit09], and a fully decentralized file system [KF06; AEHF08]. Each of these applications serve a different purpose, but each of them uses the KBR API as common denominator, which demonstrates the flexibility of the API.

3.3.2. KBR vs. Distributed Hash Tables

A KBR overlay locates data items in a fully decentralized system. A DHT stores data in such a system. It works just like its local counterpart. Given a *key* it looks up its *value*. The key space is maintained across a number of nodes. Commonly, the service includes algorithms for nodes joining and leaving the network, thus redistributing the key space among all nodes. Furthermore, caching and replicating algorithms are available to increase the systems reliability and responsiveness. The most popular academic DHT used to be the now dysfunctional OpenDHT [RCKS05] [CRR+05]. Meanwhile, commercial providers offer similar services e. g. Amazon's Dynamo [DHJ+07].

Even though many applications successfully use a DHT as underlying service, it has always been clear that a plain DHT does not suffice [RCKS05]:

> *We remain hopeful that sophisticated applications can be layered on top of a DHT service, but think that DHT services should slightly broaden their interfaces.*

Rhea encouraged the scientific community, to prefer the better abstraction of P2P services at KBR level, because a systems *endpoint operations* can require more flexibility, than just storing and retrieving data items. However, over the recent years, this required broadening of the interfaces has not happened. One result of this thesis is a library that eases the development of P2P applications, developed using the guidelines of Rhea's work, introduced in Chapter 4.

3.3.3. Iterative vs. recursive routing

Routing details are irrelevant for the functionality of a DHT. The peers could iteratively search for the peer that handles a given key. If their search results in no closer peer, the search terminates. On the other hand, they could *recursively* forward the messages to the

peer in charge. The latter has a better performance [RCKS05] and has thus become the standard approach.

The routing details are, however, relevant for a generic KBR system, because, as a consequence of the routing process, the KBR system establishes per-key aggregation trees. Messages from a network region sent to the same destination are likely to pass the same peers on their way. This property allows, for example, to implement efficient caching. If, where, and how the caching shall be performed, is a matter of the application. The KBR system determines nothing but the routing structure.

3.3.4. Kademlia

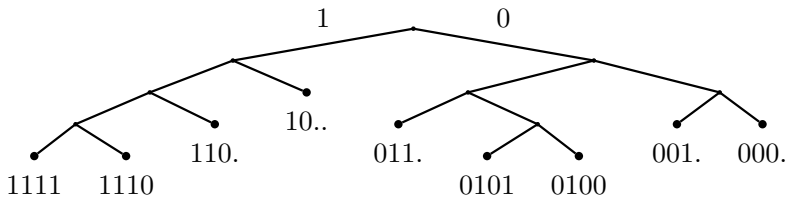

Figure 3.2.: The Kademlia binary tree with 4 bit IDs: Leaves in the tree are nodes, dots in the IDs represent hidden subtrees.

Kademlia [MM02] is a structured network, developed by Maymounkov et al. in 2002. Its structure is inspired by a binary tree, with each leaf representing a node. Congruent with the definition of structured networks in Section 3.3, each node in the tree is identified by a unique 160 bit key. Figure 3.2 illustrates the tree structure using a 4 bit ID space. Kademlia adopts a symmetric `XOR` distance metric. That metric explicitly reflects distances in the ID space with distances in the binary tree. Huge distances map to distant subtrees, for example the distance between node 0100 and node 1111 in the above figure is 1011.Small distances map to close subtrees. The authors chose Kademlia to use iterative routing. Neighbors are stored in so called buckets. A client will send a packet directly to the node that is closest to the target ID. It locates that node, by iteratively requesting closer nodes, from its currently known closest nodes in his buckets. Requests are sent to multiple nodes, so the algorithm can proceed, as soon as the first answer is received. As with all structured networks, forwarding a message to its destination is done in $O(log(n))$ hops.

3.3.5. Chord

Prior to Kademlia, in 2001 the Chord overlay was proposed at the MIT Laboratory for Computer Science [SMK+01]. In contrast to Kademlia's tree structure, the architecture of Chord resembles a ring cf. Figure 3.3. Clients joining the network will be included by inserting them between their ID-wise corresponding neighbors. Routing in Chord is clockwise. Basically a node only needs to know its successor to be able to proper route a query, based on its key, to the keywise closest node on the network. For efficiency

reasons clients in the Chord ring hold more routing information, commonly referred as "finger table", containing a list of nodes. Like most structured networks, the finger table resembles a binary search tree.

> *The i^{th} entry in the table at node n contains the identity of the first node, s that succeeds n by at least 2^{i-1} on the identifier circle [...] [SMK+01].*

In other words, Chord's routing table contains many peers with IDs close to its own ID and a decreasing number of distant peers. By using that form of finger table organisation, Chord *resolves all lookups via O(log(n)) messages*[SMK+01], because it knows its successor and many neighbors as well as distant targets.

In Figure 3.3 only $n = 3$ nodes $\{0, 1, 3\}$ exist in a network of size $m = 8$. Therefore, a query for key 2 will be clockwise routed to its closest location, in that example node 3, indicated by the respective "key" box. As the rest of the key space is empty, each "successor" entry in the finger table points to node 0, while the "start" column of the entry points rightfully to 2^{i-1}.

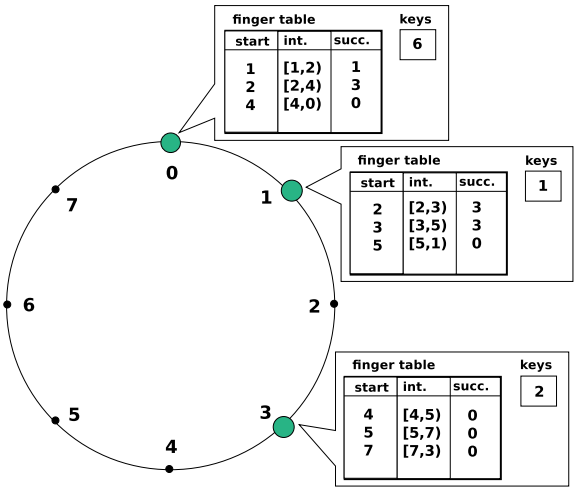

Figure 3.3.: A sample Chord network, equipped with nodes and their routing tables [SMK+01].

Chord has been granted the ACM SIGCOMM "Test of Time Paper" award, for research conducted a decade ago, *whose contents are still a vibrant and useful contribution today*.

3.3.6. Chord vs. Kademlia

Both Chord and Kademlia share a $log(n)$ lookup complexity. Kademlia's authors chose the use of iterative routing. This form of routing has benefits, but also problems cf. Section 3.3.3. Both approaches feature a tunable parameter for the routing efficiency: Altering the size of buckets is equivalent to adapting the finger tables size. Their routing system differs: *Like Pastry and unlike Chord, the XOR topology is also symmetric $(d(x, y) = d(y, x)$ for all x and y* [MM02]. In other words, a message, received via one path can be sent back via the same path. This is not the case in the Chord network, as the clockwise metric would evaluate that path as far. Therefore, Chord cannot use a potentially optimized path. This is a major drawback for Chord. On the other hand

Chord's metric is much simpler in terms of consistency. A Chord node needs connections to its right neighbor. Using that single hop the system is able to route a message slowly but correctly. Kademlia has no such constraint, and it is hard to formulate it, in terms of complexity and message overhead.

3.4. Hybrid P2P

Full distributed networks are designed with churn in mind. Therefore, they are redundant and are able to handle fail overs. However, they also have shortcomings. These include for example, membership control, privacy issues and the management of distributed applications. For example, the Servus telepathy extension [Hud08] developed at our chair faces the need to authenticate a user against its account. However, without a central user register, the user needs to claim her identify, for example using asymmetric key cryptography. As long as both users already previously exchanged public keys, they can identify each other without doubt. However, there is no straightforward solution to bootstrap that trust.

Because of these challenges, commercial focus moved away from pure P2P into server assisted- or hybrid P2P. These systems use P2P techniques to consolidate the load intensive tasks at the participating nodes. A central server handles tasks that are facilitated by having a central entity, such as billing oder authentication. For example, the previously discussed Napster system used an index server, located at a central node. But, by keeping the bandwidth intensive tasks of data transfer directly at the end user the system gained the scalability of P2P systems. The rather bandwidth preserving and seldom executed job of publishing the users available data was handled by the index server. Publishing the index in a decentral fashion is far more complicated, e. g. using a DHT requires more complex client implementations. Hence, the Napster architecture was slick and well designed, but flawed by the central component.

Today, there are numerous server assisted P2P systems in both commercial and noncommercial use. Also commercial successful applications, such as Skype, Spotify or Amazon's Dynamo use that technique.

In spite of their popularity, scalability issues imminent to central systems still exist at the central component of hybrid P2P systems. Little has been learned since the time of Napster. Two Skype downtimes illustrate that. According to the company's information, in August 2007 [VA07] the login server could apparently not handle a massive client restart triggered by a Windows update. The most recent Skype downtime in December 2010 is said to be caused from overloaded servers triggering a client bug [LR].

The most popular noncommercial Server assisted P2P protocol is BitTorrent, which will be introduced in the following.

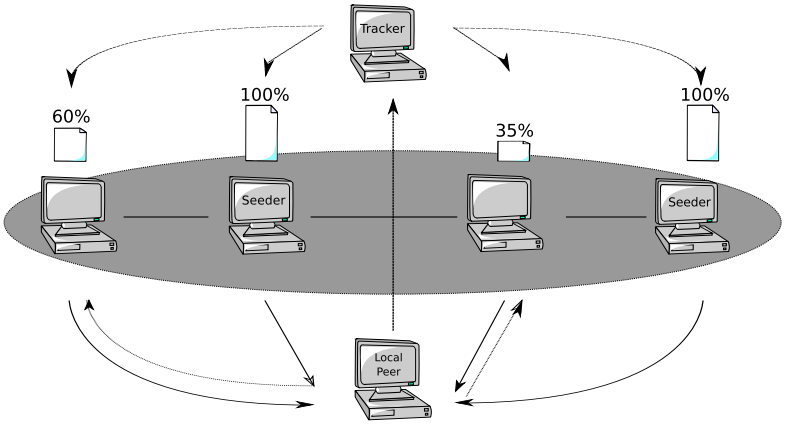

Figure 3.4.: Actors and operating mode of the BitTorrent protocol.

BitTorrent

According to Sandvine's Global Internet Phenomena report [San10, p.15], BitTorrent [Coh03] is currently the most popular file sharing protocol in the US and worldwide. It is responsible for 34.3 % of North American traffic in 2010 and has been constantly growing. BitTorrent's popularity may involve different factors, however, it stands out for its bandwidth utilization algorithm. According to [QS04] BitTorrent achieves near to optimal global download performance. The system works in the following way:

To publish a file on the network, the file's content is split up into equal sized data blocks (chunks). A metadata file describing the file and its constituting chunks is created. Every peer intending to download the published files needs this so called "torrent file". The acquisition of that file is not part of the BitTorrent protocol and is commonly provided by a web server. The metadata includes the size of a chunk, a list of each chunks Hash ID and the Address of the coordinating server (*tracker*).

The *tracker* is the system's central server component. It acts as a broker, distributing peers addresses among each other. Previous systems, like Napster provided an index server broking between participants, based on content demands. It was attacked for assisting in illegal content distribution. BitTorrent in contrast relocates the file discovery back to the users. The server is limited to maintain the list of registered clients. By design all metadata is strictly limited to the torrent file. Hence, the trackers mere function as client broker keeps it from aiding copyright infringements.

Each peer is connected to a subset of all peers, its *sworm*. Based on a variant of a *tit-for-tat* algorithm it decides which other peers gain a share of its bandwidth (unchoke). This algorithm rewards peers that cooperated, while preferring high bandwidth peers. In that way BitTorrent achieves its outstanding download utilization. New clients that have not contributed previously, may be selected at random in each fourth iteration of the algorithm. In that way, the system remains open to newcommers. Figure 3.4 sums up the different actors of the system.

The BitTorrent protocol still maintained, whereas many others are no longer developed. It is in the focus of research for nearly a decade now, for example the most recent International Conference on Peer-to-Peer Computing (P2P'11) featured several publications on BitTorrent and an exclusive track on "Bittorrent algorithms".

Vuze

A popular BitTorrent client is *Vuze* [Vuz]. The Java written software, previously named Azureus was first released in 2003. The platform independent client implements the BitTorrent protocol and provides a powerful plug-in API [Incb]. Built upon this API or into the core client Vuze's features include Vivaldi network coordinates (cf. Section 5.5), plug-ins providing DHT functionality and various front ends to the client.

Based on the large user base and the easy extendability, Vuze is popular in the scientific community. Various experiments distribute the client across PlanetLab nodes [BND10; SB09]. Ledlie et al. designed and implemented an optimization to the Vivaldi network coordinate algorithm called Pyxida, in close interaction with the Vuze development community cf. Section 5.6. The Ono plug-in from Choffnes et al. is another research plug-in that introduces a novel *implicit* approach to latency prediction into the Vuze client cf. Section 5.10.

However, Vuze's features come at a cost. In [SB09] Steiner et al. demand the noticeable lag in Vuze that is introduced by the Java based framework. That lag is reflected in the measured latencies. Therefore, they differentiate between application round trip time (ARTT) and round trip time (cf. Figure 3.5).

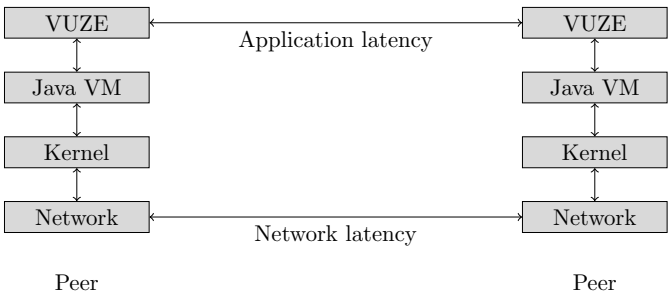

Figure 3.5.: The difference between application and network latency (cf. [SB09]).

This thesis also presents research that was partially conducted using Vuze's extension mechanism. Section 8.1.2 of this work presents the Hierarchical Vivaldi plug-in for Vuze.

4. Igor

The Internet Grid Overly Routing daemon IGOR, developed by Kutzner et al. [Kut08], laid the foundation for the present thesis. My thesis converts the IGOR system into an IGOR ecosystem. The process is twofold:

First, my thesis enhances the KBR daemon ("IGOR" or "IGOR daemon" hereupon) to take advantage of proximity information (cf. chapter Chapter 5), notably the "Hierarchical Vivaldi" algorithm introduced in Chapter 6. The development of proximity extensions not only improves the performance of the KBR daemon, but also introduces benefits for developers and researchers. Notably the extension API has seen significant changes. Furthermore, testing and verification of the software introduced simulation capabilities that reuse the IGOR daemon code seamlessly.

Second, the thesis introduces the *libdht*, which enables application developers to implement DHT access with custom endpoint operations. Previously, the IGOR daemon exported only a KBR library – libigor. Therefore, common code e. g. managing objects and their respective lifetime had to be duplicated by each application. The new library provides a unified implementation that respects Rhea's appeal of "slightly broadened interfaces", by providing a high degree of extensibility and customizability. Finally, the library offers simulation capabilities decoupled from the IGOR daemon, another strength of the employed KBR interface.

In the next section, I describe the daemon's architecture, the overlay network itself and the changes introduced in the present thesis. In the following section, I describe the extended DHT library that provides access to the KBR system.

4.1. The KBR daemon

The daemon was designed following the design principles introduced in Chapter 3. The KBR split adheres the Unix philosophy, of doing only one thing and doing it well. Hence, it is a generic KBR daemon that supports all kinds of peer-to-peer applications, called IGOR. An associated library – libigor – provides the KBR interface to the applications.

Basic routing system

IGOR employs Chord's ring topology and finger concept, customized for using a symmetric metric similar to Kademlia. Hence, messages are forwarded towards the destination key in both ways, clockwise and counterclockwise. The advantage of the symmetric metric is that reply messages can travel along the same path on which the request

messages was sent. Reusing the message's route allows the receiver to use a potentially optimized path, which the sender chose for latency or bandwidth reasons. By reusing it, the receiver does not need to compute another optimized solution. Moreover, application specific caching is easy to implement because an intermediate node is likely to receive both the request and the response.

IGOR's decision about the establishment of fingers is based on a rating algorithm. To determine the different levels of gain, the daemon includes ratings for physical proximity. The concepts of proximity based enhanced overlay creation and maintenance will receive attention in Chapter 5. In brief, the IGOR daemon uses proximity neighbor selection for the rating of potential fingers, i. e. *if* to accept a new peer. Furthermore, IGOR uses proximity route selection when forwarding a message, i. e. deciding *who* is the next hop? The required location information is calculated using an enhanced Vivaldi network coordinate system, as described in [EFF10] and Chapter 6. Peers exchange that information with a method similar to Meridian [WSS05] (cf. Section 5.9), i. e. when being in contact, peers preferably exchange information about peers that are assumed to have a low mutual latency. Other sources for determining proximity, such as AS membership, have not yet been integrated, but could easily be added to IGOR's modular design. Therefore, the implementation is open to both ISP and CDN-based oracles.

Because of the latency and caching advantages discussed in Section 3.3.3 messages are sent recursively on the overlay network. Therefore, each intermediate hop may serve as cache or handle the encapsulated payload in an application defined manner.

IGOR is implemented using TCP because it matched most feature requirements for actual connections listed in [Kut08]: Connection orientation, flow control and delay measurements. These would otherwise have required a complex mechanism on top of UDP. Therefore, Kutzner chose TCP as underlying transport medium. However, due to the software's design other algorithms may be implemented.

4.2. Socket Interface: *libigor*

The IGOR daemon is a stand-alone application. Clients connect to the daemon using the libigor library. Internally this is implemented via socket communication. Hence, even remote applications, e. g. behind a firewall, can use the daemon located in a demilitarized zone (DMZ).

libigor's API is similar to the Unix socket API providing `socket`, `bind`, `send`, `recv` and `close` functions. It mimics the `bind` system call by registering a service identifier (cf. Section 4.3) instead of ports. However, overlay communication is not a traditional end-to-end communication. Upon receiving of a message, an application may take an action depending on whether it is the destination or an intermediate hop of a message. Therefore, the IGOR daemon introduced two flags, the `upcall` and the `final` flag.

The `upcall` flag controls the flow of a message, i. e. whether it will be delivered to applications on intermediate hops. The `final` flag indicates if the node is the message's final destination. Because messages terminate at nodes with the closest ID to their destination, these two IDs will unlikely be identical. Applications that use IGOR are unaware of the routing state, which is handled in the daemon. Therefore, such an indicator is a convenient result of the decoupling of routing daemon and message handling. A received message with unset `final` flag can only appear with set `upcall` flag inside an application.

Another extension to the socket concepts is a notification mechanism for topology changes. When the neighbors in the KBR overlay change, applications may need to adapt. E. g. DHTs need to check if their stored keys need to be replicated to the new peer. The library realizes this by calling a user defined function that can be registered via the `igor_register_callback_neighborset_change` call. The user is able to register a payload, to be handed to the callback.

4.3. Services

A generic KBR system should isolate the different applications, running on its nodes. I. e. only peers that run a given application should be bothered by the respective overlay traffic. E. g. a mobile phone that joins a light-weight chat application should not have to relay heavy traffic from a video-on-demand system. On the other hand, general information that is useful for every node, e. g. peer recommendations, should be spread widely. This sharing of general information especially helps applications that have only a small user base. Being part of a large network helps them, e. g. to bootstrap quickly or to smooth stabilization in case of node churn.

IGOR addresses this issue with the concept of services [Kut08; DKF08]. Just as every node is in an overlay, each application is part of its own virtual service overlay, identified by a unique service identifier. Messages that have a specific service identifier attached, will be delivered only to nodes in that service overlay. Certainly, a peer can be part of multiple service overlays. That peer will share messages and state for all its applications along with their respective services. Thereby, IGOR efficiently optimizes the overlay network, and isolates the different applications.

Holding state for each virtual ring, increases communication as well as memory demands. However, the implemented solution mitigates the bad performance of using isolated overlays, i. e. starting a new daemon for each application on a node. Clearly, more advanced solutions e. g. ReDir [RCKS05] and DimishedChord [KR04] exist. LiteRing [DKF08] includes an analysis of each proposals complexity, which is available in Table 4.1.

	ReDiR	Isolated Overlays	DimChord	Lite-Ring
App. Inst. Join	$O(\log N \cdot \log M^*)$	$O(\log^2 N)$	$O(\log M)$	$O(1)$ or $O(\log \log N \cdot \log M^*)$
State per App. Inst.	$O(1)$	$O(\log N)$	$O(\log M)$	$O(1)$
Load Ratio	$O(1)$	$O(1)$	$O(\log M)$	$O(1)$
Routing (worst case)	$O(\log N \cdot \log M^*)$	$O(\log N)$	$O(\log M^* + \log M)$	$O(\log M^*)$

Table 4.1.: Service Separation Complexity comparison [DKF08]. Let M be the overall number of nodes, M^* be the number of nodes in the DHT and N the average number of service members.

4.4. Plug-in System

The IGOR ecosystem is the result of multiple research projects [Kut08; KCF05; KF06; AEHF08]. Therefore, the design and development of all software components including the daemon is optimized for greatest flexibility and extendability. Due to that design, proximity research that is the focus of my thesis could easily be implemented and tested. A building block of IGOR is the modular design using plug-ins. The plug-in API was intensively used and extended in this thesis.

There are four basic types of plug-ins. Each of them is responsible for a different functionality.

- Connection policy plug-in – Plug-ins of this type evaluate potential new finger connections. They evaluate both connection requests from other peers and recommendations to connect to another peer. (I. e. when peer A recommends peer B to connect to peer C.)

  When multiple policy plug-ins are loaded they decide jointly whether to accept or decline a connection. The decision is based on an additive average of all plug-in evaluation results. However, each plug-in can demand to accept or to decline a connection regardless of the evaluation. Such an override is, for example, used to always accept direct neighbors in the nodes' key space.

- Message plug-in – Plug-ins can define their own message types. The Meridian gossiping algorithm, for example, needs to exchange the peers' coordinates. Hence, every plug-in that defines its own message type must implement an according message plug-in. It determines how the messages are handled.

- Task plug-in – Task-plug-ins are required for periodic actions, for example, regular neighbor probes.

- Snoop and piggy-back plug-in – Plug-ins can snoop on the (payload) traffic. They can also piggy-back their own information onto other messages. The proximity extension, for example, attaches its coordinate estimations to messages that head to its immediate neighbors. (The latency measurement is not part of the overlay. The

daemon reads the round trip times directly from the underlying TCP/IP network stack.)

One important element of my thesis was to actually compare different network coordinate algorithms and their configurations. However, coordinates were attached static to the ID of a node. Previously, an ID (precisely a cNodeRef) of the IGOR daemon was a three tuple of the following form:

```
<Node ID> <IP:Port>+<Coordinate>
```

Using a static form for IDs is suboptimal, because it lacks extensibility and allows only one coordinate per ID. Therefore, I implemented a scalable and flexible solution.

ID add-ons A node ID identifies an overlay node uniquely. It is included in each communication between nodes. It is also sent when recommending nodes to other nodes (gossiping). However, to decide if a connection is valuable a client might need extra information. E. g. clients behind a NAT gateway are unreachable without connection reversal. To make them a priori aware of any extra information, IGOR provides a way to extend the ID information.

Extensions to the IGOR daemon can register so called *add ons* (**cNodeRefAddon**) to the IGOR ID (**cNodeRef**). Three functions provide this functionality:

```
bool HasAddon(const string &id) const;

template <class T>
  T GetAddon(const string &id) const;

void UpdateAddon(const string &id, const string &payload);
```

Each add on is identified by its unique `id` string. Add ons register and update their payload via the **UpdateAddon** call. The payload parameter contains the extra information provided by the registered plug-in. With the help of that identifier, nodes that receive an ID that has add on information can hand it to the corresponding class, if it is registered in the client. If it is not registered, or unknown, the client is nevertheless able to process all known parts of that ID. Therefore, the plug-in mechanism at ID level assists development of IGOR because not all nodes need to execute similar software versions.

Add-on data is intentionally of type `string`, hence **cNodeRef**'s most prominent function is **ToString()**. Even inside the **cNodeRef** add ons are stored as string, therefore type correctness is limited to the templated **GetAddon** call. The design rational is the absolute necessity for a slim and simple *libigor*. libigor needs to understand IDs. Hence, it links against the specific program code. However, libigor would need

to contain symbols of every add-on, if objects were not registered using their `string` representation. Hence, libigor would contain machine code only targeted for the daemon. This would result in increased library sizes and more complex linkage, without any benefit. The ID add-on kept IGOR's modular design rationale and made lightweight simulation components possible.

IGORs extended ID format is:

```
<Node ID> <IP:Port>+{<id> <payload>}*
```

In its new form each ID obligatory contains the actual ID of the node and it's transport information. Thereafter follow all registered add ons. An + sign indicates the begin of a new add on. The `id` component is the unique identifier of that add-on, as passed as the first parameter to **`UpdateAddon`**. A whitespace marks the beginning of the string representation of the add on's payload.

4.5. NAT Traversal

A shortcoming of choosing TCP as underlying basis of the IGOR daemon is connection setup in the light of network address translation (NAT) and firewalls. Due to its simplicity UDP traffic can employ a technique called "Hole Punching" or STUN [RWHM03]. However, there are reasons to ignore that UDP based workarounds:

In practice, firewalls and network address translators (NATs) obstruct the access to other peers. On the one hand firewalls implement an administrative decision, which IGOR wants to obey. So, it refrains from stealth technologies, which e. g. use HTTP or SSH ports for a P2P overlay. NATs, on the other hand, should also be considered as technical solutions to deal with address space limitations. Furthermore, the correctness constraints of the overlay require at least connections between neighbors. Therefore, some connections need to be established. So, IGOR includes all the readily available solutions for NAT traversal that do not require tampering the operating system's IP stack. Furthermore, tampering in IGOR is outside its reach:

To refrain from stealth techniques is both an ethical and technical decision: The established STUN technique is UDP based and even worse, the proposed TCP based STUNT method [GF05] requires excessive packet manipulations, while being not guaranteed to succeed. These techniques firstly could circumvent an administrative decision and secondly introduce complex privileged code paths into the system.

Therefore, one element of my thesis explored the available options to handle NAT at the overlay level cf. [Wil09]. The solution is threefold.

First the IGOR system uses the *miniupnp* library [Ber]. It provides means of port forwarding from a router to a client, using the *Universal Plug and Play* (UPnP) protocol. That facility is common to home routers and allows users to easily implement port

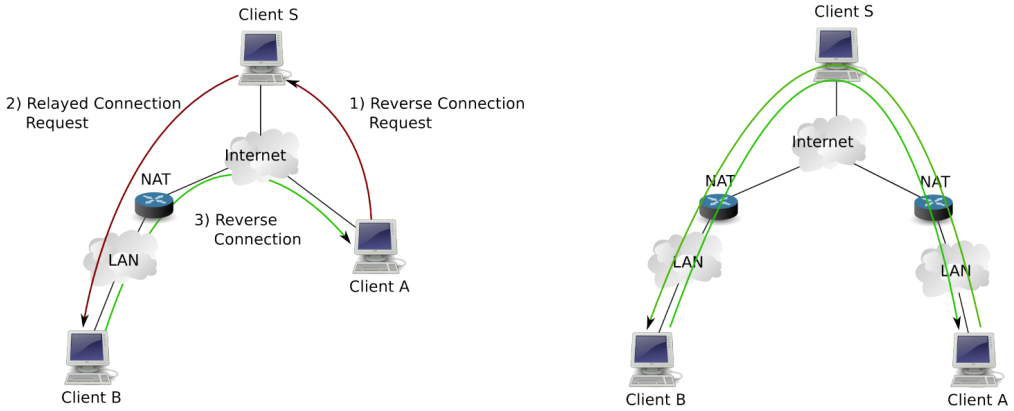

Figure 4.1.: Connection Reversal (left) and Relaying (right) as means to circumvent Network Address Translation.

forwarding. Hence, if a user enabled that functionality, she explicitly agrees to that behavior.

Furthermore, during startup, after trying to configure UPnP, nodes test their connection status by analyzing, their number of successful outgoing and incoming connections. If only one side of a connection is behind a NAT, IGOR supports connection reversal as illustrated in Figure 4.1 (left): Client node B is behind a NAT, but connected to S. As A is unable to connect directly to B, it requests S to relay its connection request to B, which in turn can now initiate the connection.

If both nodes use are behind network address translation they are unable to establish any direct communication, cf. Figure 4.1 (right). A and B need a third party S to relay their communication. The use of relay nodes is a suboptimal solution – both security-wise and traffic-wise. Hence, the relay functionality must be explicitly enabled for both the client using a relay and the server acting as relay for a connection.

As a matter of fact, the NAT state of a node represents valuable meta information, another peer could use *a priori* i. e. when learning about that peer. Hence, with the extended ID specification abilities introduced previously any peer is able to publish her detected NAT state as ID add on. Decision functions, such as the IGOR daemons connection plug-ins use the node's NAT state, to compute more accurate recommendations, by accounting the eventual overhead for NAT traversal. As a result of this thesis the IGOR daemon is able to connect and operate facing network address translation.

4.6. KBR Simulation

The design decision of separating the KBR daemon and higher level functions is also sensible for simulations. Typically, detailed and reproducible results from a realistic network simulator are crucial to study e. g. the effects of proximity enhanced routing or inter AS traffic. However, this level of detail is unimportant for higher levels. The

KBR split solved that by decoupling those higher levels from the actual overlay. Hence, by skipping everything but the KBR overlay, computing power can be spent on bigger, more realistic network topologies.

Previously, IGOR has been successfully tested on PlanetLab [PACR02], a global research network. The network is built from nodes at research institutions running the PlanetLab software. It currently consists of 1098 nodes at 530 sites [oPU]. A downside of that service is inherent to its research foundation. PlanetLab nodes are homogeneous machines because the participating organizations are commonly well connected and bandwidth rich. However, the Internet is fundamental heterogeneous, with various different and varying latency and bandwidth setups.

Hence, the KBR daemon was tested in medium scale homogeneous networks. Furthermore, these tests were not as reproducible, as if running in a simulation environment. Functionality had been historically tested in local tiny setups only. These tests were easier to reproduce, but still not identical.

That limited testing hampered the development of correct and realistic tested algorithms. Furthermore, simulating at the network level allows the development and evaluation of proximity predicting algorithms in a controlled environment. Therefore, a simulation component for the whole IGOR ecosystem emerged as one goal of the present thesis.

I evaluated different simulation framework, focusing on two criteria: First, the simulation component needs to be able to seamlessly integrate into the IGOR software, to maximize the coverage of code that could be tested. It should be able to support my research in proximity optimized overlays, therefore it should be able to realistically simulate a TCP stack. The evaluation resulted in two frameworks that matched these requirements. I will introduce both in the following.

4.6.1. PRIME SSF

In [DV09] we analyzed the KBR overlay daemon using an earlier version of the simulation component, based on Prime SSF [LMVH07]. Prime SSF implements the Scalable Simulation Framework (SSF) [CLL+99], which serves as description of an interface rather than an implementation. SSF consists of that API and the Domain Modeling Language (DML), a description for networks of all kinds. Together with the SSF Network extension, which contributes interfaces for network protocols like TCP, SSF forms a simulation framework for distributed applications. There are various Java implementations and the PRIME SSF implementation written in C++ available. As the KBR daemon is written in C++, PRIME SSF, in the 3.0.1 version from August 2008 was chosen as the basis for IGOR's simulation component.

The PRIME simulator makes heavy use of annotations to the application's source code. Methods and variables are to be annotated, to help the simulator with the discovery of variables, that need to be saved on context switch and which functions cause a context switch. After an initial source - to - source translation, using the *p4* preprocessor special

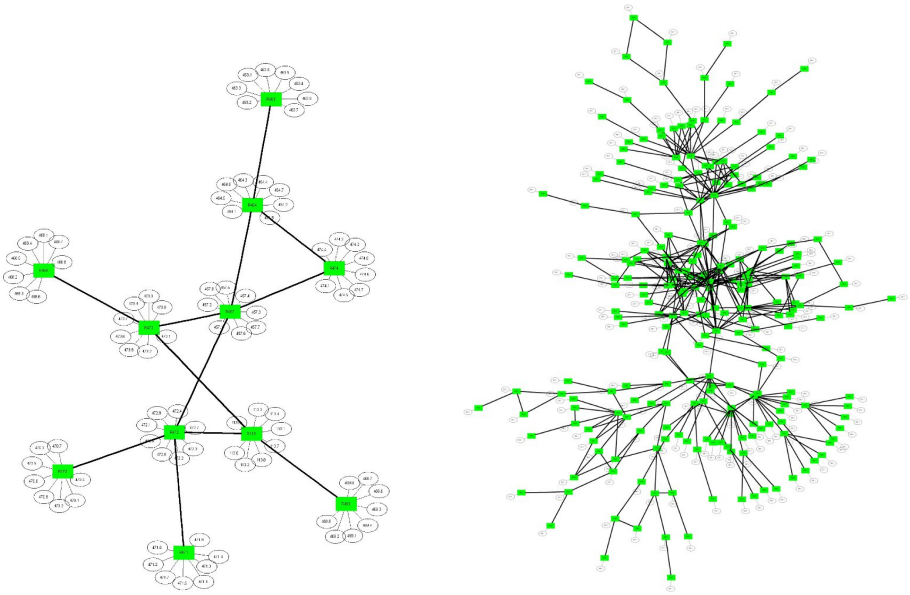

Figure 4.2.: Simulated Indian (11 routers, 8 hosts/router) and Nacamar (242 routers, 1 host/router) network topologies.

makefiles compile a monolithic program. Supplied with several DMZ files, describing hosts and networks, the simulation starts.

In contrast to synchronous UNIX socket programming, the simulator is event based. Hence, calls to blocking functions result in "context switches" to other simulated nodes, the result of a blocking function is passed via a callback. Upon a "context switch" the simulator processes its queue until the simulation time elapses to the completion of this call.

First, we analyzed the structural correctness of the overlay. We used input data from the RocketFuel [SMW02] project to create a realistic copy of the Nacamar/Tiscali provider network and an Indian provider's network. Figure 4.2 illustrates the topologies. Furthermore, the same code ran on PlanetLab using 118 nodes. We analyzed the number of messages of the stabilization protocol and the time until the network is stable i. e. until the virtual ring is closed. In both settings, messages were sent to random destinations, with a small set of neighbors that ensured overlay paths with multiple hops. After verifying the correctness of our results and the congruency of the simulation and the PlanetLab results, we performed initial measurements comparing it to a Gnutella based system.

The development of the PRIME simulation component was a first step to the simulation component envisioned as goal of this thesis. SSFNet, however, left room for improvement, for example the lack of distributed simulation capabilities, the static routing model and defects in the software. Due to these problems, and the obstructive code changes introduced to IGOR, SSFNet was replaced by a pure *OMNeT++* implementation. The structure of the simulator fits much more to the existing design of IGOR.

4.6.2. *OMNeT++*

OMNeT++ is an extensible, modular, component-based C++ simulation library and framework [Com]. It supports a broad range of networks, such as sensor or wireless ad-hoc networks. Although calling itself a framework, *OMNeT++* [Var01] is a comfortable simulator that even comes with a graphical user interface. Equipped with the INET packet, which just like SSFNet, implements the Internet Protocol Stack, *OMNeT++* matches and supersedes SSF's features.

The simulator is event based, however the integration into the IGOR KBR daemon was less intrusive than the SSF port and resulted in smaller and more stable simulator code. The class `cIgorOmnetTransport` and `cIgorInetTransport` realize the respective transports. Both are subclasses of `cIgorTransport` that provides the abstract message passing APIs. This way, the same code that runs on the real network in the KBR daemon can be tested in a simulated environment without modifications.

After switching to *OMNeT++*, I reproduced results from a measurement study on PlanetLab in [EFF09]. We provided IGOR with topology information based on RTTs from that study. During the measurement, 120 PlanetLab nodes ran the IGOR software. We were able to simulate all nodes on an eight core machine in twice the measurement's real time. Our findings indicate that using the local simulation component is a valid alternative to large scale testing in a complex distributed environment.

The major insights gained from both simulator ports are:

- The importance of simulating *running code*, hence testing as much code as possible inside a simulator
- The importance of reproducible measurements and behavior

The development of P2P applications benefits substantially by the incorporation of simulations at the earliest development stages. I consider the ease of using *OMNeT++* is seen as important factor of its success.

That experience gained from two simulator ports allowed a third development. To explore the behavior of the Vivaldi network coordinate algorithm and my proposal Hierarchical Vivaldi a third simulator was built with a focus on large simulations. This was accomplished by the drastic reduction of features from the simulator cf. Chapter 7. Neither a real transport protocol nor a real P2P application is simulated. The simulator provides an event queue that manages RTTs and invokes the network coordinate algorithm upon its objects. From an architectural point of view, IGOR's modular design drastically reduced the development time. IGOR's network coordinate elements were already self contained and easily combined into a thin library *libvivaldi* that provided the implementation for the simulator's core task.

On the DHT level, libdht offer a functionality that is similar to Free Pastry [RD01], as described in Section 4.7.

4.7. Application Interface: *libdht*

I based libdht's design on the results of Chawathe et al. [CRR+05] cf. Chapter 3. In particular, I aimed at a generic library that provides more than just the usual DHT functionality. The library should be flexible enough to support all kinds of application requirements, and it should be focused enough to allow the efficient sharing of code.

The most common requirement is of course the general put/get interface for storing key/value pairs. Moreover, applications need to be able to perform endpoint operations that are more complex than just storage. For example, they need the flexibility to control *what* data is stored and *how* it is stored. They also need to control the forwarding process, for example, to suppress the further forwarding of a message.

The principal architecture was chosen and designed by Kutzner in [Kut08]. The decision to extend and extract the architecture into a library comes from the understanding that the lack of a public extensible library has unnecessarily hampered the further development of efficient P2P systems and their widespread use. Neither should an optional library "get in the way" of the developer, nor should it be only marginal necessary and especially not be overloaded with features. Figure 4.3 illustrates libdht's position in the ecosystem. Application developers could use it to enrich their application, they can abandon that option, or they can use it in parallel.

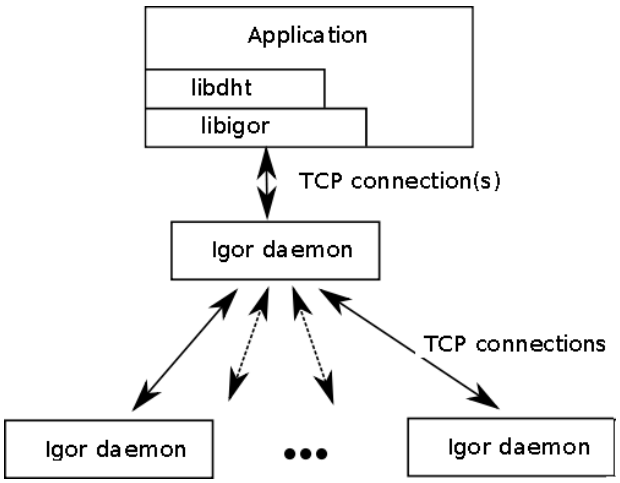

Figure 4.3.: libdht and libigor in the IGOR Ecosystem.

The library inherits the architecture from its predecessors: the IgorFs file system and the VIDEGOR video recorder [KCF05; KF06; Kut08]. Common to both predecessors is the modular approach to the architectural design, illustrated in Figure 4.4. Each type of functionality is implemented in a separate module, inherited from `cModule`. These Modules communicate via an IPC based messaging system, using the following interface:

```
void cModule::SendMessage(
        shared_ptr< cMessage const > message,
        cModule *destination,
        posix_time::time_duration relative_when
);
```

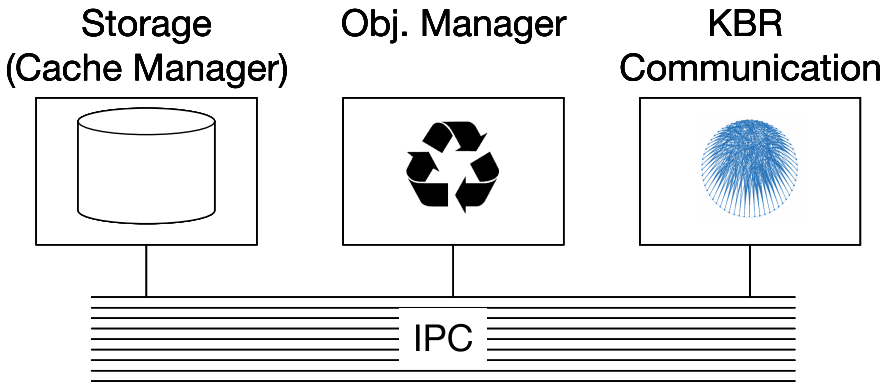

Figure 4.4.: The module concept of libdht.

```
virtual void cModule::HandleMessage (
        shared_ptr< cMessage const > message
);
```

Upon construction, modules can specify the type of messages, they handle. Upon receiving, their `HandleMessage` method is executed. Commonly a worker thread is spawned to handle the new message, to avoid blocking the main message loop.

Figure 4.4 includes the library's core components. The "KBR Communication" module is the responsible module for easing the communication with the underlying KBR daemon. It multiplexes messages of different types, received by the lower levels of the software stack and sends them to the corresponding modules. The "Object Manager" serves as central component for application developers. By handing object references to that module, the library stores the modules in the last illustrated module,the "Cache Manager". From the latter storage module, the Object manager retrieves the application provided objects and handles their life cycle, by republishing them timely. The latter module also serializes the cache content upon library unload, to preserve caches across application runtime. The underlying IPC bus handles the message loop and calls the `HandleMessage` functions.

Functionality is not limited to the three modules. Application developers are free to develop their own `cModule` derived extensions. In [Wei09] we explored the concepts of BitTorrent's cooperation engine and their applicability to DHT concepts. We found the concept generic enough to be applied operate distributed in a DHT. Therefore, we created a custom `Upload-` and `DownloadManager` module in libdht that enables the joint download of a resource from multiple peers. Together with a third `cSwarm` module, application developers can enrich their applications with swarming functions.

Each module can be individually re-implemented and replaced. However, modules can even be customized at a more fine grained level.

In libdht, an application can provide *filters* that run whenever a peer receives or sends a message. A filter can check the received data. It can process the data and trigger its further dissemination; or it can suppress the message from being further forwarded towards the final destination. Thereby, an application can efficiently aggregate and cache information.

Filters do not only apply to received messages, but also to outgoing cache elements. Thereby, the system can transform, suppress, or validate the data before it is sent. A filter for incoming messages implements the following API:

```
class cCacheInputFilter {
        virtual bool Run (shared_ptr< cMessage const > message);
        virtual string HandledNameSpace ();
}
```

Its counterpart, the filter for outgoing data operates on elements, exported from the object storage:

```
class cCacheOutputFilter {
        virtual bool Run(
                cCacheElement const *inputElement,
                cCacheElement **outputElement
        );
        virtual string HandledNameSpace ();
}
```

The provided functions are located at the heart of the system. While implementing a complete module might be cumbersome, filters allow extensive changes in minimal lines of code. Upon registration and initialization of the filter, the system automatically checks the result of the **`HandledNameSpace`** call. Depending on type of data stored in the cache or read from the cache, different filters get activated. The filter's namespace selects these types.

I illustrate the effect of that design with an example. Consider, for example, an application layer multicast application, say a chat service or an audio conference tool. Here, a prospective multicast group member sends a join request towards a given key. Typically, this key will identify the group.

Upon receiving such a request, the application's input filter checks if it has already registered another downstream peer. If so, it adds the new requesting peer. Otherwise, it

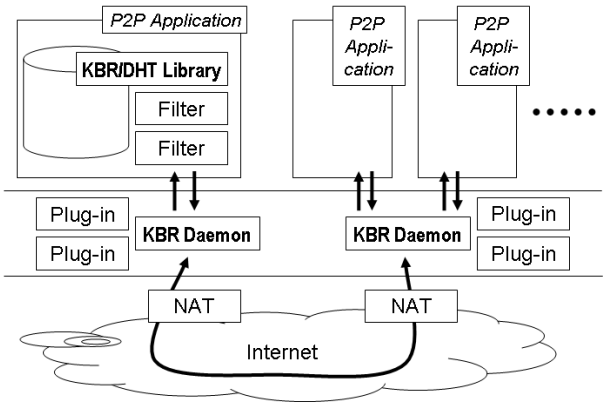

Figure 4.5.: IGOR Application Interface: Architecture Overview.

creates an according record, adds the requesting peer, and issues its own join request for that group. Multicast traffic is forwarded towards the said group key and to the locally registered peers (except for the peer that the particular traffic came from).

libdht Simulation

Besides the relative ease of developing P2P applications with the libdht library, it also supports application developers with its simulation capabilities. Using the built-in simulation capabilities, the students at our chair's lab were, able to simulate a multicast application built with this library on a network with 200 peers and 10 different groups on their desktop PCs. They were able to quickly track down the errors in their implementation, and they were easily able to simulate both gracefully and ungracefully leaving peers. Thereby, they could build an application that proved to be robust after real-world deployment. As the live system and the simulation both use the same code base, they

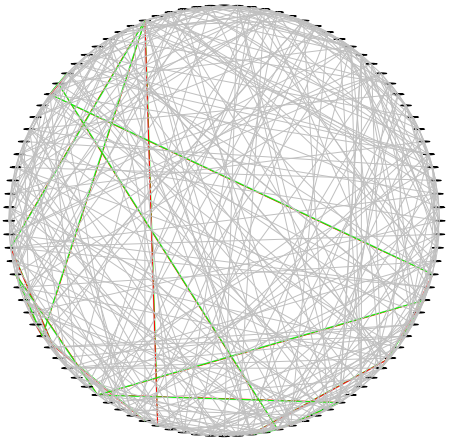

Figure 4.6.: Multicast traffic from 100 nodes created in the built-in simulator.

were able to reuse all their instrumentation for the performance analysis after deployment.

The key to that simulator lies in the KBR split cf. Section 3.3.1. Because *libigor* is separated from all routing logic, by communicating only via `route` calls with the underlying daemon, the library can operate on all kinds of different KBR proposals. And because the application library *libdht* is split into modules that communicate via IPC, the component communicating with the KBR daemon can naturally be replaced with another implementation. Hence, I implemented a fake transport. Instead of calling into a library or contacting a daemon the reimplemented `cFakeIgorModule` communicates with other instances of itself via IPC. I implemented the clockwise routing metric from Chord [SMK+01], however, any other routing metric is possible. The interface to the simulator is currently limited, but serves as proof of concept. It is able to simulate node churn and can delay messages at will.

Finally Figure 4.5 illustrates the combination of all parts to an ecosystem. Messages issued from applications, pass the KBR/DHT library's filter system, where they may be transformed, queued, cached or forwarded to the KBR daemon. Each message that enters or leaves the daemon is passed to plug-ins at the KBR level. The service plug-in inside the daemon takes care of using only overlay hops that provide the same service. Other plug-ins decide, which overlay path the message shall travel dealing with connection specific attributes, including NAT and round trip times (RTT). Finally, the message is delivered, thereby traveling through the stack in the opposite direction.

In [Sch10] we implemented our previous example of a multicast add-on for the KBR/DHT library. This was done to prove the point of the great extensibility and modularity of the library. During the development, no changes to the library were necessary. Using the built-in simulator a network of 100 nodes was able to join 10 multicast groups concurrently on a commodity hardware system (cf. Figure 4.6).

5. Proximity Enhancement Research

This thesis is focused on proximity enhancements for creating latency optimized overlay networks. Chapter 2 introduced the sources of latency and how it accumulates. Although these reasons may be known, developing a good prediction algorithm is challenging. The prediction of RTTs in a decentralized fashion is even more an elaborate art. This chapter will introduce the most widespread algorithms and their approaches to reach the goal of an accurate prediction.

5.1. Overview

Early works on P2P overlays completely ignored the topology of the underlying network. But completely ignoring the underlying routing paths hampered the efficiency of the overlay routing. To illustrate the problem, consider two nodes A and B, located in Europe that do not share a direct overlay connection. In a network of n participating nodes, the path between both nodes consists of up to $log(n)$ hops. Any of these hops could be located overseas, which adds the round trip time between Europe and North America, approximately 115 milliseconds, to the path. In a worst case scenario, the path includes multiple bounces between nodes in the US and Europe. As a result, P2P overlays waste the Internet service providers' (ISPs) bandwidth and annoy the users with high latencies. The proposed optimizations for overlay networks are based on a simple observation: Because the overlay totally abstracts from the underlying network, routing paths may become unnecessarily inefficient.

The obvious solution would be to route the message to the destination, using an optimal routing path. Clearly, the meaning of optimal depends on the context: An ISP might be interested in keeping traffic local, or prefer his peerings. Users care for low latencies and high bandwidth. This thesis focuses on prediction of latency optimal paths, therefore both "optimal" and "near" hereafter mean latency close. The problem is: How to deduce that optimal routing path? Sampling all possible choices results in massive overhead. Flooding a message guarantees to find the latency optimal path, but is inefficient and might congest the network. Prediction seems to be impossible because of [Pax97], but Paxson only concluded that the behavior is unpredictable. In this chapter, the reader will be introduced to algorithms that predict the actual RTT based on recent observations. By observing the system, predicting is indeed possible.

There are several proposals how a peer can find proximate peers without having to sample a lot of candidates. Vivaldi [DCKM04] employs network coordinates, i. e.

each peer is assigned a synthetic coordinate so that the computed distance of two peers reflects their expected latency. Elser et al. [EFF10] and Agarwal et al. [AL09] propose optimizations to this algorithm, but stick with its coordinate based model. Meridian [WSS05] is also based on the idea that peers can be embedded into a metric space, but avoids explicit coordinates and rather uses the measured latencies directly. Ono [CB08] uses content distribution networks as oracles to determine the proximity of peers. Other authors propose that the ISPs offer proximity recommendations to P2P systems [AFS07; XYK+08].

Using any of these prediction techniques overlays are proximity aware and thus much more efficient [CLY+09; LOH+10]. They allow peers to determine their latency adjacent neighbors. So they can preferably attach to these selected peers and thereby create efficient overlay structures (cf. Figure 5.8). Similarly, when routing a message, these peers preferably forward them to low latency peers.

The problem of optimizing a path or choosing the optimal neighbor is reduced to the problem of latency prediction. The fundamental advantage is this prediction property: A priori sampling all nodes is minimum bound to the first answer, only if sampling happens in parallel. In contrast, prediction involves no previous communication to obtain latency characteristics, hence no additional time loss emerges.

A focus of current research is the problem how to improve the prediction quality. The de facto standard today is computing and assigning synthetic coordinates to Internet hosts with regard to their latency characteristics. So far I have introduced the proposals briefly, to create an overview. In the next sections I introduce each in detail, to impart the reader a sound understanding of the state of the art.

5.2. Definitions

Prior to introducing proximity prediction algorithms, I start with a formal definition of the problems that motivate this thesis:

The Peer Selection Problem

> *The optimal peer selection problem is to select, from the set of peers that have the desired object, the subset of peers and download rates that minimizes cost. [AKR+05].*

This problem is closely related to two other problems:

Proximity Route Selection PRS can be employed everywhere in overlay networks, where the forwarding rules and metrics allow to choose from more than one entry in the routing table. Pastry was the first system to allow such flexibility. In contrast initially both Chord and Kademlia used a strict metric, which was subsequently softened, when the need for proximity enhancement became evident. From the set of all possible next

hops PRS first selects all next hops allowed by the routing metric, i. e. all next hops that decrease the distance to the destination in the identifier space. All next hops in this set are then evaluated using a second metric, e. g. latency. The message is then forwarded to the next hop where the second metric is optimal.

PRS can also create a drawback, as a path chosen to optimize the second metric can contain more hops than the path chosen by the primary metric, which can lead to a longer transmission time. This potential drawback cannot be avoided, as only the next hop in the path can be optimized. Global optimization would require global knowledge. PRS requires a choice from two not necessarily congruent recommendations. The IGOR KBR daemon reflects that stress ratio between different metrics, by basing routing decisions on a joint decision of connection plug-ins (cf. Chapter 4).

Proximity Neighbor Selection While a truly decentralized P2P application cannot rely on global knowledge to optimize its routes it can approach the problem in another fashion. Instead of introducing a secondary metric, violating or bending the primary one, the system could try to run the primary metric only on preselected optimal members. Therefore PNS optimizes the connection table. In that way the secondary (proximity) metric decides on neighbor selection and applies optimization at this point. This is possible because commonly P2P system exchange neighbor information (*gossiping*), to optimize their routing tables. The proximity metric can be integrated at that point, selecting latency close entries from the received information, discarding others. The optimization process might even involve replacing existing connections.

To sum it up, the *Peer Selection Problem* is not only a problem of selecting from whom to retrieve an object. If an overlay's construction or routing is guided by latency considerations, the performance increases. Hence the Peer selection problem is a design problem, that improves the overlay as a whole. By designing an overlay based on proximity, the intercontinental overlay path from this chapter's introductory example could be avoided.

Embedding The focus of this thesis is on the analysis of network coordinates. Algorithms that assign coordinates to reflect RTT distances rely on an approximately exact layout of nodes on a plane. In contrast, similarity based algorithms like Ono cf. Section 5.10, express distances using implicit measures. The process creating a layout on a planar surface is called a *embedding*, a concept well known from topological graph theory. Each node can be interpreted as a vertex $v \in G$ in the graph of all nodes G, leading to the following formal definition:

> *Let $G = (V, E)$ be a simple, undirected graph, with the weight function $f : E \to \mathbb{R}^+$. An embedding (ϕ, R, d) of a graph G is given by a non empty set R, a function $d : R \times R \to \mathbb{R}^+$ and an embedding function $\phi : V \to R$. The pair (R, d) is the*

surface or space into which G is embedded. G together with the function f form the underlying graph [Lau04].

The distance function d maps two nodes to their respective distances. For example, on the Euclidean plane, the distance function is the norm of the vector between the two points. In contrast distance in a hyperbolic space is bent towards the origin of that surface.

An actual distance is computed using the node's positions on the surface G, which is assigned to each node in V by ϕ. For convenience reasons hereafter the representation $\phi(v_i)$ of node v_i on the surface (R, d) is called the node's coordinate x_i. Furthermore, the weight function is commonly replaced by the *latency matrix* $L = (l_{ij})\ L \in \mathbb{R}^{x \times n}$, that contains the latency l_{ij} between all nodes i and j.

The quality of an embedding is determined by the degree the distance function d reflects the function f. To measure that quality of an embedding (ϕ, R, d) an error function E is commonly defined. For example, the central Vivaldi algorithm described in Section 5.5 minimizes the following error function:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{n} [d(\phi(v_i), \phi(v_j)) - f(v_i, v_j)]^2, \tag{5.1}$$

That specific error function in the above equation computes the sum of the quadratic absolute embedding errors. Furthermore, different forms of embedding algorithms exist cf. [ST04]:

- **All pair (AP)** Embed all nodes ($\frac{n \times (n-1)}{2}$ distance pairs) at once.
- **Two phase (TP)** Embed a small subset of landmark nodes ($\frac{l \times (l-1)}{2}$ distance pairs) in a first phase. All other nodes, embed their distances using these landmarks in a second phase.
- **Random + Neighbors (RN)** Embed respective to the neighborhood N of a node and randomly selected distinct nodes.

After introducing the most important definitions, the remaining chapters will focus on proximity predicting algorithms.

5.3. Early Latency Prediction

Internet Distance Map Service One of the earliest proposals for latency prediction is the Internet Distance Map Service (IDMaps) project [FJJ+01]. Francis et al. envisioned a service, which provides latency information *with a delay and overhead less than those of the gains achieved by using the service.* Their proposed solution settled on landmark

servers, also called *tracers*. The authors chose these servers because *a large number of hosts making independent and frequent measurements could have a severe impact on the Internet*. In other words: The number of measurements need to be conducted in a system should not be quadratic bound to the number of nodes $|N|$ in that system ($|N| \rightarrow |N|^2$ measurements). Instead, only each tracer $t \in T$ with $|T| \ll |N|$ needs to determine that latency. IDMaps distance computation is illustrated in Figure 5.1: The distance d_1 between two nodes A, B sums up from the distance of both nodes to their nearest tracer ($d_A + d_B$) *plus* the distance between both tracers. Hence, the distance between node A and B is $d_A + d_{t12} + d_B$. Clearly, IDMaps is an early two phase embedding algorithm.

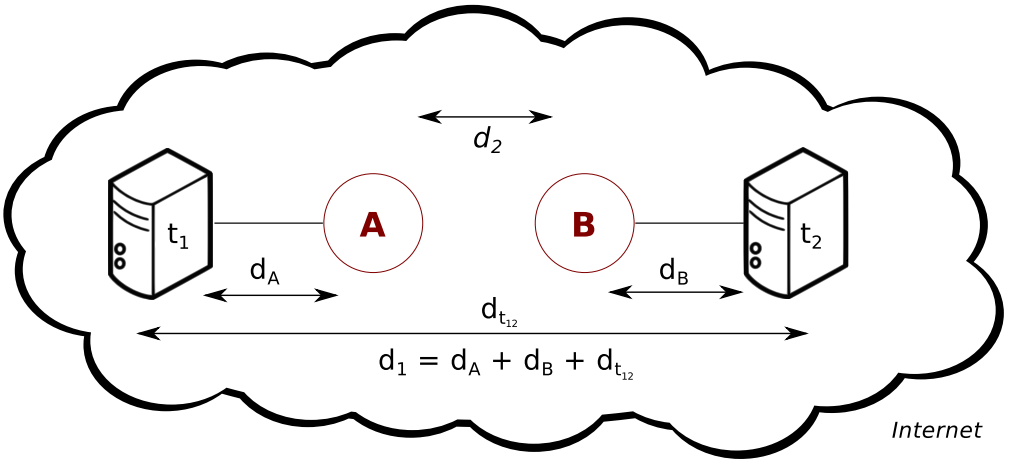

Figure 5.1.: IDMaps lack of client position information induces errors at close peers. IDMaps computes d_2 from both AP distances plus the tracer distance.

To maintain the correctness of the distance information, tracers perform periodic measurements among all $|T|$ nodes. These measurements are compressed using a *t*-spanner algorithm, to limit the size of the exchanged data. Nodes are maintained in groups, based on their address prefix (AP e. g. 10.1.1.0/24) at their corresponding tracer. Distances between tracers and APs are learned while operating the network. Upon receiving an initial request from an unknown AP, the contacted tracer advertises itself as the nearest tracer to that AP. The tracker publishes this information across the tracer network. If other tracers measure and find shorter latencies, they take the position of nearest tracer. The system is designed as a client/server solution, hence end hosts can query dedicated servers for distance estimations.

Global network positioning One problem of IDMaps comes from a simple observation illustrated in Figure 5.1. The actual distance between node A and B is d_2 not d_1. A distance computed with IDMaps is not able to reflect peer positions. Hence, although both peers may be close, the computed distance will still reflect the distance of both peers to their respective tracers, plus the distance between those tracers. Similar problems

exist with nodes sharing a tracer. As one of the IDMaps authors put it: *IDMaps is blind to position* [ST04].

Therefore, Global Network Positioning (GNP) [NZ02] pioneered with the concept of network coordinates. Nodes exchange the computed coordinates. These coordinates increase the accuracy of distance prediction and eliminate IDMaps blindness for position. Furthermore, the exchanged coordinates allow end hosts to independently compute their respective distance. Therefore, a server that is necessary in IDMaps is redundant in GNP. That independence not only saves the latency of a query to a central server, but also improves reliability, by eliminating a potential bottleneck.

To achieve stable coordinates, the authors propose a two phase algorithm. $\mathcal{D}+1$ landmark servers $\mathcal{L}$ initially compute their coordinates of dimension $\mathcal{D}$. Coordinates $x_1 \dots x_N$ are computed by minimizing the following equation:

$$f_{obj1}(x_1 \dots x_N) = \sum_{i,j|i>j}^{n} \varepsilon(d(x_i, x_j), l_{ij}) \tag{5.2}$$

Where $\varepsilon()$ is an error function e. g. computing the absolute displacement between real and computed latency. The authors propose to solve resulting multidimensional global minimization problem using solvers, such as the Downhill Simplex Algorithm.

After a stabilization period, clients can measure their RTT to landmark servers and compute their coordinates based on these measurements. Subsequently, landmarks adapt to changes inside their respective networks, using continuous measurements.

Vivaldi adopted the idea of network coordinates, but replaced the fixed set of landmarks with its spring approximation model. This makes Vivaldi a self-organizing system in the sense that it is independent of a landmark infrastructure, as we will see in Section 5.5.

5.4. Big Bang Theory

The concept of Network Coordinates using Euclidean surface pioneered in the GNP system. To improve the graph positioning, which is responsible for efficient and accurate node placement, Shavitt et al. proposed the *Big Bang Simulation (BBS)* algorithm in [ST04]. BBS uses force fields to compute each node's kinetic energy. The authors found improved performance and accuracy of their embedding algorithm, compared to other methods such as Downhill Simplex. They emphasize the algorithms ability to avoid suboptimal embeddings:

> *the kinetic energy accumulated by the moving particles enables them to escape the local minima. [ST04]*

The idea of force fields that actively push particles apart allows all nodes to be initially placed at the same point, hence the name Big Bang Simulation. The force is a function of

the embedding error of the system, the error function. Its antagonism is a friction force. The former pushes particles apart, increasing their velocity, the latter slows them down. The algorithm tries to arrive at a minimal force and friction configuration at a perfect embedding.

The system is divided into four phases. In the first phase, the error function $E^{\langle 1 \rangle}$ is computed from the absolute error between the actual and computed distance between particles. The authors describe the force in that phase similar to a spring network.

$$E^{\langle 1 \rangle} = \sum_{i,j}^{n} (\|v_i - v_j\| - l_{ij})^2$$

That model is the basis of the Vivaldi algorithm, hence the next section will explain it in-depthh. The next phase uses the relative directional error between both values.

$$E^{\langle 2 \rangle} = \sum_{i,j}^{n} \frac{E_{ij}^{\langle 1 \rangle}}{\min(\|v_i - v_j\|, l_{ij})}$$

In the last two phases, the directional relative error increasingly contributes to the error function. Hence, the system is increasingly sensitive to embedding errors.

$$\begin{aligned} E^{\langle 3 \rangle} &= \sum_{i,j}^{n} \exp^{(E_{ij}^{\langle 2 \rangle})^{\frac{3}{4}}} - 1 \\ E^{\langle 4 \rangle} &= \sum_{i,j}^{n} \exp^{(E_{ij}^{\langle 2 \rangle})} - 1 \end{aligned}$$

The force computation requires distance information between all particles, which requires a full meshed P2P network. That hampers its usability for truly distributed applications. Furthermore, the observed performance during simulations varied. Synthetic Internet topologies, based on power-law graphs benefited from hyperbolic embedding. Topologies gathered from measurements on the Internet did not exhibit an advantage over traditional embedding schemes [ST08].

5.5. Vivaldi

The Vivaldi algorithm (Vivaldi henceforth) developed by Robert Cox et al. [CDK+03] is a force based algorithm, for creating a network embedding. The algorithm incorporates inter alia research from the GNP algorithm, by using network coordinates and the Big Bang Simulation, which pioneered with the idea of force fields. Vivaldi replaces particles and force fields with a spring embedder cf. Figure 5.2. Initially all coordinates are placed at a single origin, similar to BBS. Because Vivaldi skips the different phases from BBS

and creates at the same time, remarkably good embeddings, the algorithm is currently an established procedure of predicting Internet latency between arbitrary Internet hosts.

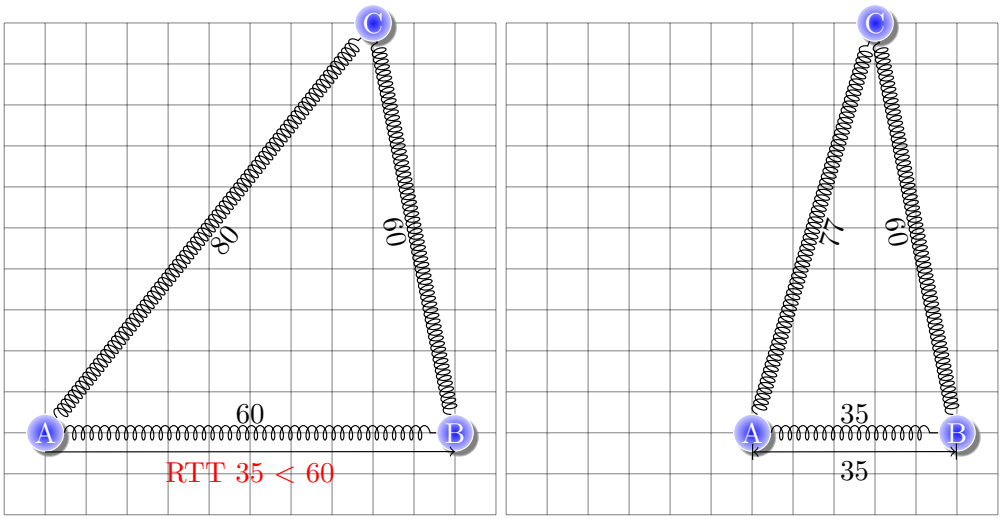

Figure 5.2.: Vivaldi's spring concept illustrated: Node A and B supposedly 60 ms apart (left). A RTT measure determines 35 ms. Hence, nodes adjust to 35 ms (right) after a number of rounds under ideal circumstances.

Embedding: The learning problem, of assigning accurate coordinates to hosts, is solved by simulating a physical spring network, as illustrated by Figure 5.2. A virtual spring is placed between each pair of nodes. The spring's rest length is the latency between the node pair. Its current length is set to the Euclidean distance between them. This displacement is identical to the prediction error of the coordinates. These simulated forces cause the correct adjustment of each node's network coordinates, starting from an initial placement. Each iteration of the algorithm should minimize the prediction error, which represents the potential energy of the system.

5.5.1. Central Vivaldi

To compare the performance of their decentralized algorithm, Dabek et al. proposed both a central and decentralized variant of Vivaldi. This section briefly introduces the central Vivaldi variant. Similar to Big Bang Simulation, the algorithm requires knowledge of the latency matrix $L = (l_{ij}) \in \mathbb{R}^{n \times n}$ that provides RTT information for each pair of nodes. In contrast to BBS, Vivaldi sticks to the same error and embedding functions, abandoning special functions for different phases. The algorithm is an *all pair* algorithm and does not use landmarks to restrict the number of computations from $\frac{n \times (n-1)}{2}$. The error function minimizes the following equation:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{n} (l_{ij} - \|x_i - x_j\|) \tag{5.3}$$

Each iteration of the algorithm between two nodes i and j, takes their coordinates x_i, x_j and their respective round trip time RTT $= l_{ij} \in L$ as input. The algorithm's result is an aggregated force vector $\vec{F}$ that adjusts the coordinate of node i. The vector $\vec{F}$ pushes the node's coordinate away from nodes that wrongfully seemed close and moves the coordinate into the direction of nodes that were placed too far away.

Algorithm 1 describes the algorithm in pseudo code.

Algorithm 1: The Central Vivaldi Algorithm [CDK+03].

Input:
x_i: local coordinate of node i
x_j: remote coordinate of node j
l_{ij}: RTT from node i to node j
Constants:
t: damping constant
Output:
x_i: updated coordinate of node i

```
1  function central-vivaldi(RTT, x_i, x_j)
2  begin
3      for i = 1 to n do
4          F ← 0
5          for j = 1 to n do
6              e ← l_ij − ‖x_i − x_j‖
7              F ← F + e ×unit(x_i − x_j)
8          x_i ← x_i + t × F
9      return x_i
10 end
```

Central Vivaldi operates on a fixed v_i for each pair $(v_i, v_j) \in E$ in the following way. The function $unit(x_i - x_j)$ computes the norm vector of length 1, pointing from x_j to x_i, which is multiplied with the absolute displacement between the computed and real round trip time. The resulting displacement vector is aggregated in the force vector $\vec{F}$. Each individual displacement contributes equally to $\vec{F}$. Finally the resulting force vector is added to x_i after damping $\vec{F}$ with the constant t, which moves the coordinate into the predominant direction of all forces acting upon it. Hereby t regulates the coordinates reaction to these forces. A value $t < 1$ will provoke a smaller adjustment, $t = 1$ applies the original force and $t > 1$ provokes a strong reaction.

5.5.2. Dynamic Vivaldi

In order to to increase the embedding quality of coordinates, the central Vivaldi algorithm introduced in the previous section, as well as the Big Bang Simulation, or similar proposals are all-pair algorithms. Therefore, they needed connections between all nodes. Vivaldi in contrast restricts the algorithm's input to a node's neighbors and creates a *Random + Neighbors (RN)* embedding. All nodes cooperatively contribute their local knowledge to the embedding. Restricting to a node's neighbors makes Vivaldi applicable for a truly distributed operation.

However, two problems of the Random + Neighbors embedding need to be addressed, in order to mitigate the smaller number of nodes that is in this embedding, compared to an all pair embedding.

First, the speed of convergence is an important factor. Nodes need to be able to quickly reach an accurate position in the embedding. A quick convergence towards an accurate position is not only beneficial for the local node, but also for the whole network. Therefore, the algorithm should produce a strong reaction to the shifting force. But a strong reaction is only initially desirable, because as the accuracy of a node's coordinate rises, a strong reaction will build up oscillations and hampers the convergence towards stable coordinates. Because of Internet's imminent fluctuation of RTTs or transitory phenomenons e. g. route flaps accurate coordinates should only produce a small reaction to such disturbances. In BBS, Shavitt et al. introduced a friction force, to anticipate these oscillations.

Second, a node that receives coordinates from another node lacks information about the quality of that information. On the one hand, it could be an accurate embedding that allows the other node to predict RTTs rather well. On the other hand, it could be a low quality coordinate for various reasons, such as the following: The other node might have joined the network only recently, or it might have experienced route flaps, which might decrease the quality of its embedding.

If a peer adapts its coordinates to nodes with low quality coordinates, its own embedding quality might decrease. Hence, in contrast to the central Vivaldi algorithm, a peer should not adapt to forces from all nodes equally in the same way. It should rather weight the force vector that alters its own coordinate with a factor that measures the other embedding's quality.

To overcome both problems Vivaldi introduces a local error component e attached to each coordinate. Using the error e_j from the remote node j a node i is able to measure the quality of $j's$ coordinate x_j. Thus, in the first line of Algorithm 2 computes an error weight w. That weight is a sliding mean of both the local error of the node e_i and the remote error e_j:

$$w = \frac{e_i}{e_i + e_j}$$

If the local error e_i is low, the computed weight will favor the own position, leading to a small reaction. On the other hand, if the remote error is low, w ensures a strong movement towards the other node's position. Therefore, w reflects the above mentioned concerns. If a node joined the network recently, it will quickly adopt towards an increase of embedding quality. If the node already has a satisfying coordinate, it will react with smaller adjustments. Figure 5.3 illustrates the median error of nodes in a simulated environment. The setup and data sets of that figure are described in Chapter 7. After an initial phase of high errors, all nodes jointly improve their coordinates towards a high quality embedding, reflected by a moderate local error of 0.2.

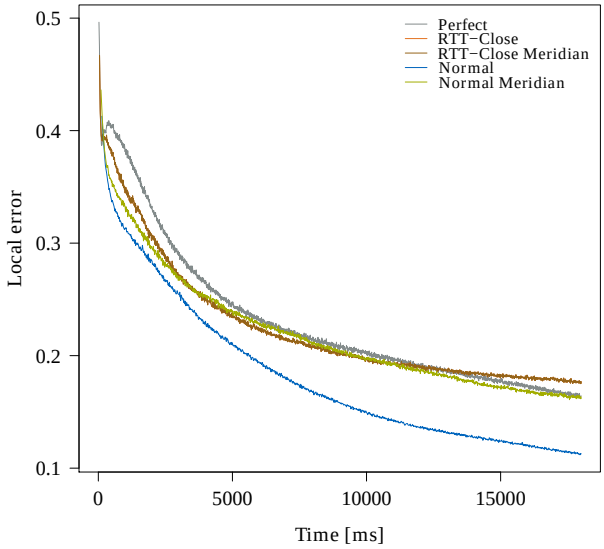

Figure 5.3.: The local error of a simulated network.

The computation of coordinates is described in algorithm 2. It starts by computing the already described weight. Hereafter the relative error of the measurement, the displacement relative to the actual latency is computed. By using two constants c_e and c_c, the algorithm first computes its updated local error, and computes the damping factor δ, which controls the actual adjustment of the node's coordinates. A uniform vector from x_i to x_j is set to δ in each dimension and finally updates the coordinates.

Chapter 7 presents an excessive simulation study, reaching beyond our original paper in [EFF09]. In the publication, we focused on the influence of various parameters, including c_e and c_e and the choice of a latency matrix. Here I present further insight, into on the choice of neighbors and their influence in the embedding quality.

Another contribution of Vivaldi is the proposal of a height vector that expresses an additional distance to the target. That distance was introduced to improve the overall

Algorithm 2: The Vivaldi Algorithm [CDK+03].

Input:
x_i: local coordinate of node i
x_j: remote coordinate of node j
l_{ij}: RTT from node i to node j
e_i: error estimation of node i
e_j: error estimation of node j

Constants:
c_c: coordinate adjustment constant
c_e: error adjustment constant

Output:
x_i: updated coordinate of node i
e_i: updated error estimation of node i

function Vivaldi($l_{ij}, x_i, x_j, e_i, e_j$)
begin
  $w \leftarrow \frac{e_i}{e_i+e_j}$
  $e_s \leftarrow \frac{|\|x_i-x_j\|-l_{ij}|}{l_{ij}}$
  $e_i \leftarrow e_s \times c_e \times w + e_i \times (1 - c_e \times w)$
  $\delta \leftarrow c_c \times w$
  $x \leftarrow x + \delta \times (rtt - \|x_i - x_j\|) \times unit(x_i - x_j)$
  return (x_i, e_i)
end

performance of Vivaldi and should explicitly capture the access network latency from the end user hosts to their provider. Coordinates are adjusted in the following way. Each n dimensional coordinate $x \in \mathbb{R}^n$ gains an additional height component x_{n+1} also called x_h. Therefore, x is now $x \in \mathbb{R}^n \times \mathbb{R}$. However, the height component is not an additional dimension, but is treated separately during the computation. When subtracting or adding two coordinates both height components $x_h + y_h$ are added to the to that result. The norm function therefore has the following form:

$$\|x\| := \sqrt{\sum_{i=1}^{n} x_i} + x_h$$

The distance function includes both the updated vector norm and subtraction functions:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} + x_h + y_h.$$

Besides capturing the access network latency, Chapter 7 introduces another beneficial property of height vectors. They are better at embedding non Euclidean data, hence data that violates the triangle inequality. That violation degrades the quality of an embedding, as I will introduce in the next section.

5.5.3. Triangle Violations

Embedding errors are inevitable for all network coordinate systems, if the input latency matrix, is already flawed with violations of the triangle inequality equation (TIV). These violations are common in real-world networks, but cannot be modeled in an Euclidean space with positive height displacement [Lau04]. This affects the performance of network coordinate systems significantly. Using simulations of several Vivaldi variants with up to 32 dimensions (cf. Chapter 7), I could show that the quality of the embedding even drops when using more than 10 dimensions.

Chapter 2 introduced the reasons for these violations. In this thesis I present simulation studies using latency data available from several scientific publications. That data sets are the King-Blog, MIT-King, Meridian and Azureus latency sets (cf. Section 7.2). As a preparation for the simulation studies, I analyze how common TIVs are in each of these sets. Not only the existence of TIVs but also the degree of them, deserves analysis. In conformance with a definition by Ledlie et al. [LGS07] I define ρ_r as measure for TIV violations.

$$\rho_r = \min_{z_1,z_2,\ldots,z_r} \left(\frac{l_{xz_1} + l_{z_1z_2} + \cdots + l_{z_ry}}{l_{xy}} \right)$$

where l_{xy} is the measured latency between nodes x and y. Nodes z_i represent intermediate nodes on the path between x and y. $\rho_1 \in \mathbb{R}^+$ is the ratio of the shortest 1-hop indirect path via z_1 to the direct path from node x to node y. A value of $\rho_1 < 1$ indicates that there is a path that violates the triangle inequality. ρ_∞ reveals the existence of a shorter path including an infinite number of intermediate hops z_i.

Figure 5.4 shows that paths in the Internet commonly violate the triangle inequality. All data sets exhibit not only single-hop violations of the triangle inequality, but also multi-hop violations. Hence, all Vivaldi variants that have been proposed so far face the fundamental problem of embedding Internet latencies into a metric plane.

The impact of the data's non metric characteristics receives attention in Section 7.4. That section will use ρ to verify an algorithms' capability to cope with non metric data.

5.6. Pyxida

Research around Vivaldi [LGS07] proved its scalability, but also disclosed some problems with the system. The original algorithm had been picked up and included into the popular Vuze BitTorrent client (cf. Section 3.4). Using collected data from that deployment, Ledlie et al. revealed problems with Vivaldi in real world situations [LGS07].

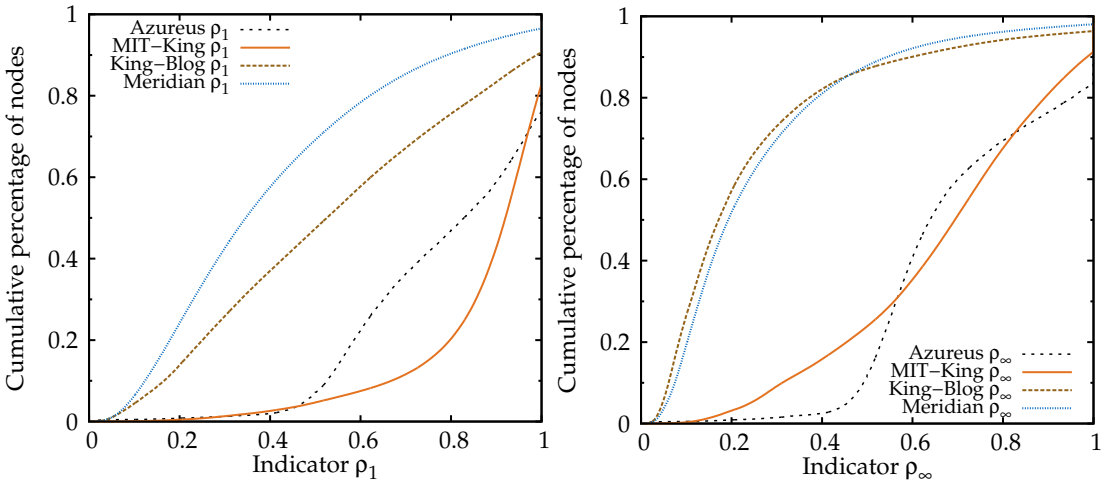

Figure 5.4.: Single-hop (left) and multi-hop (right) violations of the triangle inequality.

First of all, a node does not contact its peers equally often. According to Ledlie et al. the resulting imbalance has a negative impact on the global optimization process. Secondly, real world influences inevitably create spikes in the RTT measurements, which according to the authors of [LGS07] distort the coordinates of an otherwise stable system massively.

Ledlie et al. proposed two improvements to the Vivaldi algorithm that compensate for these problems: A low pass RTT filer based on mean values to include only plausible RTT values. Therefore, each node should keep a RTT history of all contacted nodes and judge new RTT measurements based on the their previous measurements. Furthermore, their proposed revised algorithm addresses the imbalanced measurement frequencies. The new algorithm migrates Vivaldi to a round based system. Algorithm 3 summarizes Pyxida in pseudo code. Instead of computing the force vector $\vec{F}$ from a single measurement, each node keeps a list of its most recently used peers (*recent neighbor set*, *R*). It contains those peers with whom a node ran the Vivaldi algorithm at most a time e_t ago. All these measurements, one for each node in R, jointly contribute to $\vec{F}$ of the current measurement. The older a measurement, the less it contributes to the coordinate corrections (line 11), Ledlie et al. call that property "neighbor decay". The force vector $\vec{F}$, which changes the peers' coordinates, is then modified as follows

$$\tilde{F} = \sum_{j=1}^{N} F_j \cdot \frac{a_{max} - a_j}{\sum_{i=0}^{n-1} a_{max} - a_i} \tag{5.4}$$

where $a_{max} \leq e_t$ is the maximum age of an entry in the recent neighbor set, a_k is the age of the kth entry, and $\vec{F}_k$ is the force pushing in the direction of that entry.

The most important contribution of Pyxida algorithm is that it does not only take the most recent measurement into account, but that many measurements jointly contribute to the coordinate adjustment. This greatly reduces spikes and other fluctuations. It is

Algorithm 3: The Pyxida Algorithm [LGS07].

Input:
x: local coordinate of node
$Y = \{y_1, y_2, \ldots, y_k\}$: coordinate of nodes from the k sized neighbor set R
$RTT = \{rtt_1, rtt_2, \ldots, rtt_k\}$: RTT to nodes in the neighbor set
$A = \{a_1, a_2, \ldots, a_k\}$: time of last contact with all nodes in the neighbor set
Constants:
t: constant damping adjustment of local coordinate, similar to Vivaldi's c_c
Output:
x: updated coordinate of local node

```
1  function Pyxida(x, Y, R, A, t)
2  begin
3      s ← 0
4      F ← 0
5      a_max ← max{a_1, a_2, ..., a_k}
6      for i = 1 to k do
7          s ← s + a_max − a_k
8      for i = 1 to k do
9          e ← rtt_i − ‖x − y_k‖
10         F_i ← e × unit(x, y_k)
11         F ← F + F_i × (a_max − a_i)/s
12     return x ← x + t × F
13 end
```

also smoothes the of nodes that have a high measurement frequency compared to nodes with a lower frequency.

The effect is that Pyxida filters and smooths the RTT measurements before they are entered into the Euclidean model. The authors found out that a four dimensional Euclidean space equipped with a height vector provides a robust latency estimation.

Ledlie's proposal was included into the Vuze client and evaluated. However, the Vuze authors later removed Pyxida from the client because its improvements did not justify its resource demands. In [EFF09] we reaffirm these claims, as we found, that in practice, Vivaldi and Pyxida do not differ much in their performance. In particular, both algorithms were not able to produce coordinates that led to good overlay topologies. We attribute this disappointing result to the way in which both algorithms handle embedding errors.

5.7. Htrae

Htrae [AL09] is another variant of Vivaldi, fine tuning it in several ways. Agrawald and Lorch discovered that the actual performance of Vivaldi is influenced by the initial coordinates a node chooses and the movement of the coordinate system.

A badly chosen initial position of a node could not only result in bad performance in the starting phase of the algorithm, but could also converge to a suboptimal solution in the long run, as discovered by Shavitt et al. [ST04]. Commonly, a node initializes its coordinates at random upon the first contact with another node. Instead, Htrae uses public available data to map the peer's IP address to a real world Global Positioning System (GPS) based coordinate. Hereby, the coordinates are likely to rest at a sensible, low-error position from the start of the algorithm. Pyxida, Vivaldi, and BBS need to place their coordinates randomly or at a virtual starting coordinate, as they lack that information. Htrae's coordinates change during the algorithm, as they adapt to the actually measured RTT values. Hence, a direct relation between a node's position on the surface and a "real" geographic position does not exist.

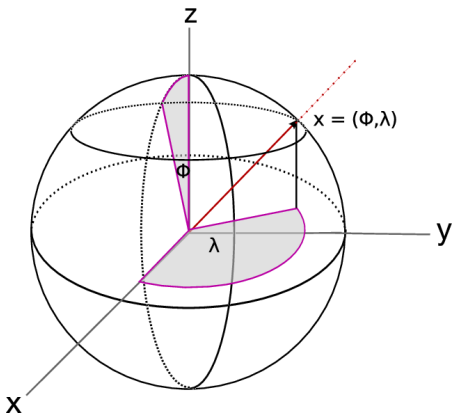

Figure 5.5.: Spherical Coordinates. Angle ϕ and λ determine the position of x.

Using the Microsoft XBox Network as testbed, the authors reassured their claim that a good initial choice for the coordinates improves Vivaldi's convergence. Inside the XBox Network, Htrae was used to select a local server for nodes that requested partners for multiplayer online games. To increase convergence speed, updates of the algorithm are symmetric, which means that a node sends its updated coordinate immediately to its communication partner.

Another proposed change to the algorithm is the use of spherical instead of Euclidean coordinates cf. Figure 5.5. A spherical coordinate $x_N = (\phi_N, \lambda_N)$ of node N consists of two angles, the latitude angle ϕ_N and the longitude angle λ_N. A distance from the coordinates origin, the radius r, which is optional to spherical coordinates is omitted by the authors.

The updated computation of Vivaldi in spherical coordinates is:

$$
\begin{aligned}
r &\leftarrow 1 - c_c w_s(\parallel \vec{x}_A - \vec{x}_B \parallel - l_{AB}) / \parallel \vec{x}_A - \vec{x}_B \parallel \\
d &\leftarrow cos^{-1}\left[cos(\phi_A)sin(\phi_B) + sin(\phi_A)sin(\phi_B)cos(\lambda_B - \lambda_A)\right] \\
\gamma &\leftarrow tan^{-1}\left[\frac{sin(\phi_B)sin(\phi_A)sin(\lambda_B - \lambda_A)}{cos(\phi_A) - cos(d)cos(\phi_B)}\right] \\
\phi_A &\leftarrow cos^{-1}\left[cos(rd)cos(\phi_B) + sin(rd)sin(\phi_B)cos(\gamma)\right] \\
\beta &\leftarrow tan^{-1}\left[\frac{sin(\phi_B)sin(d)sin(\gamma)}{cos(rd) - cos(\phi_A)cos(\phi_B)}\right] \\
\lambda_A &\leftarrow \lambda_B - \beta
\end{aligned}
$$

The algorithm resembles Vivaldi, with minor adjustments for spherical coordinates.

The authors claim that this change eliminates the drift problem. Drifting is a movement of the whole system in the coordinate space, common to network coordinate algorithms. They attribute the low drift and the good performance of the algorithm to both their bootstrapping changes and the use of spherical coordinates. The initial claim from the Vivaldi paper [CDK+03] was that other coordinate spaces, including spheres have limited effect. In [Fö09] we conducted a corresponding evaluation, which reassured Dabek's claim. The different findings of Lorch's publication on the one hand and Cox or our publication on the other can be explained in the experimental setup. Both [CDK+03] and [Fö09] did not investigate the coordinate system's possible mitigation on the drift problem and lacked GPS based initialisation.

However, the use of Geo location databases might even yield a performance penalty, as Poese et al. [PUK+11] found out, that

> *Geolocation databases can claim country-level accuracy, but certainly not city-level.*

Therefore only nodes that are equipped with an actual GPS device might benefit from their proposals. Even under those ideal circumstances, the use of 3 dimensional GPS data lacks the information that two nodes at close physical locations might be connected to different providers and could expose different latency behavior. Finally I found that reusing previously computed coordinates is sufficient, (cf. Section 8.2.1).

Other embeddings for Vivaldi were proposed, such as a hyperbolic space. In [LS08] the authors experienced mixed results with good performance for nearby peers. However the hyperbolic prediction underestimated the distant peers, therefore the authors propose a mixed heuristic ThreshHyperbolic, which swithces between Euclidean and hyperbolic coordinates.

5.8. PeerWise

In the preceding sections coordinates appear as one of several concepts to achieve latency predictions. In contrast, Section 2.2.1 introduced the PeerWise [LBL+09] algorithm to detect routing shortcuts. The algorithm's triangle inequality violation detection is based on the Vivaldi algorithm. At the heart of that detection lies the observation, that Vivaldi is based on a metric space and should be able to embed other nodes with low error. Hence, if a nodes has high embedding errors to another node, it is likely that a violation in the metric space, a TIV is the cause. The goal of PeerWise is to set up a network of detour routes, allowing packets to travel faster than the direct route.

The system tries to ensure mutual incentive. A node should not need to forward packets for two other nodes, if it doesn't benefit itself. The algorithm that ensures that global fairness in PeerWise is described in the following:

If a node A discovers a shorter path to B via another node C using gossiping or searching, it requests a detour from C. That request includes merchandise in the form of neighboring nodes of nodes A that have a high embedding error. Hence, the requested

node may pick a detour via A, which provides mutual benefit. This agreement is called a "peering" in PeerWise's terminology.

The choice of the optimal peering depends on different variables. Those include the proximity of other nodes which yield the best results in simulations, or the respective embedding error. Nodes with high proximity are likely to exploit diverse AS paths.

In an experiment conducted at PlanetLab, the authors verified their findings. 42 % of nodes saw a decrease in download times, compared to the direct path. Therefore, the authors concluded, that:

> *[...] the embedding error in network coordinate systems indicates detour routes [LBL+09]*

PeerWise is an innovative use of Vivaldi far from its original application.

5.9. Meridian

Meridian is a framework for node selection based on network location properties. It offers a multi hop search for a closest node where each hop reduces the distance to the target. Meridian guarantees to find exactly or nearly the closest node [WSS05]. It also guarantees that no hop is farther from the target than the previous.

The Meridian framework creates an overlay network based on three components:

First each node maintains a ring-structure that classifies its neighboring nodes into rings, depending on their latency distance from that node. Each ring consists of $k = m \times O(logN)$ members. N is the number of nodes participating *in total* in the Meridian algorithm. m is the number of rings. The rings size grows exponentially with growing latency-distance cf. Figure 5.6. The i-th ring has an inner radius $r_i = \alpha s^{i-1}$ and an outer radius $R_i = \alpha s^i$ for $i > 0$, where α is a constant, s a multiplicative increase factor and $r_0 = 0, R_0 = \alpha$. To mitigate churn, Meridian also maintains an unspecified number of member candidates.

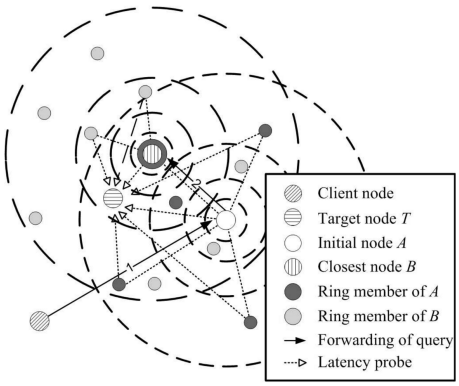

Figure 5.6.: Meridian closest node discovery [WSS05]. Client contacts node A, which forwards the request to B.

Second, a ring membership management component refreshes and improves the ring's characteristics. The selection of the ring members is done in a way, supposed to fill the ring with geographically distributed nodes. To refresh that information each ring member periodically measures its latency distance to each other member.

Third, a gossip-based node discovery protocol provides new candidates for the rings. Each node A sends a single member of each ring to a randomly selected node B. Node B can place these new nodes in its ring structure, after doing latency measurements.

Finding a closest node to v in a Meridian overlay, works in a similar way to DHT key based routing cf. Figure 5.6. While a KBR system uses ID distances as metric, Meridian replaces that metric with latency distances. Each hop A measures its latency to the target T. That node picks a set of nodes from its latency-corresponding ring structure and requests them to report their distance to T. If one of the remote nodes returns a smaller latency, than the local achieves, the algorithm restarts at that node. Therefore, each hop will find a node, nearer to the query's origin. If no close hop can be determined, the algorithm terminates.

Meridian's accuracy in finding the closest node is unique for a distributed algorithm, however, the repeated direct measurements during search make it unfeasible for a real world P2P algorithm. The principle of gossiping closest nodes to other nodes, however, is sensible and has been used in simulations introduced in the next chapter.

5.10. Ono

Choffnes et al. proposed Ono [CB08], which relies on a DNS based Content Delivery Network to create locality. In Section 2.3.3 I introduced CDNs. They typically work by replying to DNS queries with the replica server that is latency closest to the requesting peer. Ono creates AS level granular locality by resolving a vector of DNS names. The entries of that vector point to CDN managed sites from Akamai and Limelight. Throughout this section, I use the example vector, proposed in the original paper [CB08, table 2]:

DNS name	CDN	Description
e100.g.akamaiedge.net	Akamai	Air Asia (South-East Pacific)
a1921.g.akamai.net	Akamai	CNN.com (US news site)
a245.g.akamai.net	Akamai	LeMonde.com (French news site)
a20.g.akamai.net	Akamai	Fox News (US news site)
wdig.vo.llnwd.net	Limelight	ABC Streaming Video (US TV)
a1756.g.akamai.net	Akamai	Popular website

These entries resolve to different IP addresses, depending on the IP address or AS location of the inquiring peer. As the CDNs returns the nearest replica server to each client, the resolved vector will reflect a nodes position in the network similar to a fingerprint. For demonstration purposes I resolved the example server from four different German ASes in Figure 5.7.

Because larger caches may be distributed across a number of machines, Ono uses /24 address blocks, just like Address Prefixes (APs) in IDMaps. The algorithm runs

e100.g.akamaiedge.net	a1921.g.akamai.net	a245.g.akamai.net	a20.g.akamai.net	wdig.vo.llnwd.net	a1756.g.akamai.net
AS 31334 - KABELDEUTSCHLAND-AS Kabel Deutschland Breitband Service GmbH					
95.100.64	92.123.72	92.123.72	92.123.72	87.248.217	92.123.72
AS 3320 - DTAG Deutsche Telekom AG					
88.221.136	195.145.147	195.145.147	195.145.147	87.248.217	195.145.147
AS 24940 - HETZNER-AS Hetzner Online AG RZ					
95.100.128	92.123.72	92.123.72	92.123.72	87.248.217	92.123.72
AS 12816 - MWN-AS Leibniz-Rechenzentrum Muenchen					
184.85.144	212.201.100	212.201.100	212.201.100	87.248.217	212.201.100

Figure 5.7.: An example Ono vector resolved in four different AS.

periodically, remembering each result vector. From this data it computes a *ratio map* μ_α, which reflects the ratio of how often a hostname α resolves to particular replica r_n. E. g.

$$\mu_{\text{e100.g.akamaiedge.net}} = \langle (95.100.64, \mathbf{0.1}), (88.221.136, \mathbf{0.9}) \rangle$$

represents a peer that was directed to the IP `88.221.136` when resolving the address `e100.g.akamaiedge.net` in 9 of 10 algorithm runs.

The actual peer selection works by assuming that peers that receive similar results from the CDN have a low mutual RTT, e. g. they might be in the network of the same Internet service provider (ISP). The similarity is computed by treating peer a's ratio maps as vector and computing a score between a and b using the following formula, yielding a value of $[0, 1]$:

$$cos\_sim(a, b) = \frac{\sum_{i \in I_a} (\mu_{a,i} \cdot \mu_{b,i})}{\sqrt{\sum_{i \in I_a} \mu_{a,i}^2 \cdot \sum_{i \in I_b} \mu_{b,i}^2}}$$

Where I_α is the set of replica servers peer α has been redirected to. If the result is above a certain threshold (currently 0.15) Ono prefers these peers. The result does not express a real distance, Ono uses implicit distance estimations.

Returning to the previous example I assume to be in the algorithm's first round, for the sake of clarity. I. e. each ratio map contains only one tuple with 100% redirection to the sole replica. Therefore the respective scores from the peer in AS 31334 are:

AS	Ono score
AS 3320 - DTAG Deutsche Telekom AG	0.167
AS 24940 - HETZNER-AS Hetzner Online AG RZ	*0.640*
AS 12816 - MWN-AS Leibniz-Rechenzentrum Muenchen	0.167

Hence the peer from AS 24940 is recommended. Ono is implemented as a plug-in for Vuze (cf. Section 3.4) with 1,316,388 installations since 2008. The authors found that using their technique 30 % of all recommendations lie in the same AS. They conclude that Ono's mission to reduce AS hops is accomplished.

Ono's functionality is similar to GNP, but replaces the RTT measurements to the landmark servers with the CDN queries.

5.11. Sequoia

The Sequoia tree approach by Ramasubramanian [RMK+09] uses the Internet's hierarchical organisation, to represent it in a hierarchical data structure. The algorithm maps end nodes on the Internet in a hierarchical tree like structure. No intermediate nodes are represented, yet they are replaced by "virtual routers" that are pure virtual entities. The path from the (measuring) root node to a leaf sums up, by the edge weight of each virtual router along the way. The actual latency is the sum of the complete path.

In a P2P network it is impractical for each peer to create that path. Furthermore the strong server centric focus of this project hampers adoption for a distributed system. Nevertheless the study's focus on hierarchical data structures deserves attention in this thesis. The authors used a 4PC-ϵ measure, which quantifies a given structure's resemblance of a tree metric on a $[0, 1]$ scale. They studied several different latency data sets on their embed ability. Their results indicate, that the Internet's hierarchical structure can indeed be represented by a hierarchical structure.

5.12. ISP assisted Oracle services

All previously introduced solutions focus on deducing topology information about a node's home and adjacent networks. One of the most obvious solutions, relying on ISP provided information is more complex than it seems initially.

As mentioned in Chapter 2 with the expansion and rise of commercial providers, topology information is considered a business secret. Therefore ISPs are in a stalemate position. They have strong incentives to keep traffic local inside their AS, to avoid traffic costs. However, they will not provide peering and topology information to their customers, for them to optimize their requests. Quite the contrary, the Internet provider Comcast shaped the traffic of their customers to save bandwidth in 2004 (cf. Section 2.3.6).

A possible solution to this dilemma is to let the ISP influence the peer selection process, while still leaving their customers in control. Two possible solutions will be introduced in the following.

5.12.1. Oracle

Aggarwal et al. [AFS07] introduce the idea of an oracle that pursues a similar goal as Ono. Instead of relying on a CDN, the oracle is an ISP-operated recommendation server. The server is provided with a list of candidate nodes, which the server will return sorted to the client. The available sorting options differ depending on the target nodes' locations. If the destination is outside of the home AS, the list may be sorted based on number of hops between the source and destination autonomous systems or the distance according to interior gateway protocol weights.

If the destinations are inside the own AS, the ISP has more information about the candidate peers, hence the recommendations may be sorted by the geographic information, which may resolve to city or even point of presence (PoP) level. The ISP is also aware of performance levels, hence the list may be sorted using performance information, such as delay or bandwidth. Finally the infrastructure of the provider provides information about link congestion or router queues. Using the Oracle's reply, a client can select the best peer, without measuring manually.

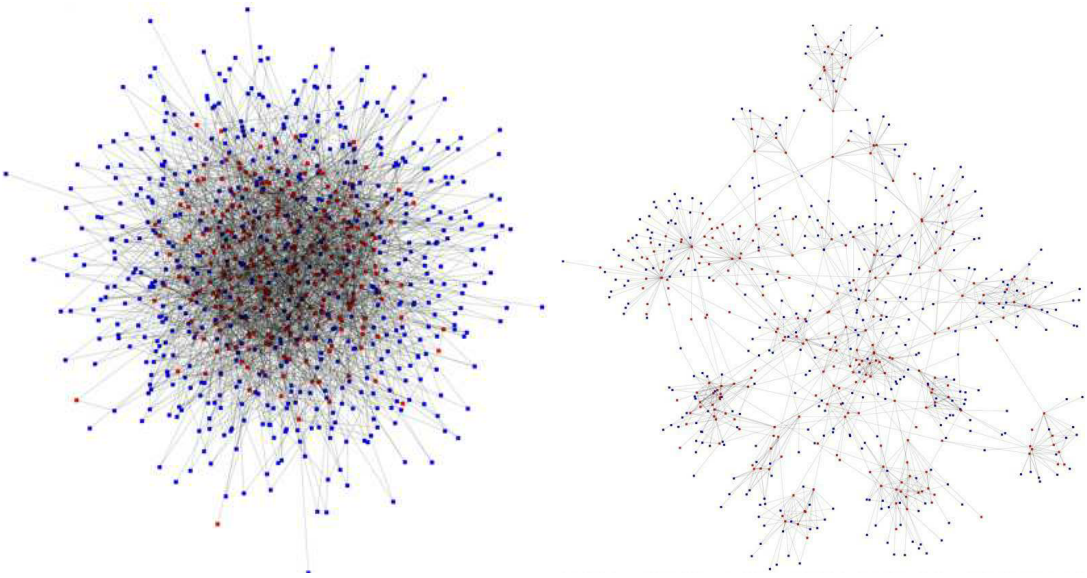

Figure 5.8.: A simulated Gnutella network. Left: No proximity enhancements. Right: Enhanced by the Oracle service [AFS07].

In Figure 5.8 the authors illustrate the difference between an optimized Gnutella network in Figure 5.8(b) and an unoptimized Figure 5.8(a). As one would expect, the unoptimized Gnutella network resembles chaos. The optimized network is the result of Aggarwal's algorithm, simulated in SSFNet. As each Gnutella client connects only to ISP recommended clients, only local connections exist. Clearly, the network reflects the underlying AS structure, which explains the perfect structure, the image on the right

exposes. While initially an optimization for the Gnutella network, the system is flexible and can serve clients when selecting CDNs [PFA+10].

One possible critic of the approach is a potential violation of the principle of "Net neutrality" [Cro07]. An ISP could sort the candidate nodes list biased by its economic interest. He could e. g. prefer nodes with a slower, but cheaper access link. A far worse problem is the users need to disclose information to a untrusted third party.

5.12.2. Proactive Provider Assistance for P2P (P4P)

A closely related solution is the P4P approach of Haiyong Xie et al. [XYK+08]. The system is designed with a focus on both side's privacy. The key observation of their system is a reluctance on both the users and the ISP's side to disclose information. P4P provides an interface between networks and applications for exchanging:

- static network policy
- P4P distances
- network capabilities

P4P lays a strong focus on privacy. Therefore the system never communicates factual data. All information exchange centers around pDistances, which describe the current costs an application, or the network experiences during a communication. These costs may reflect OSPF ranked distances, bandwidth or routing constraints. The *iTracker* is the communication endpoint between both ISP and application. In contrast to the Oracle, an iTracker is designed to communicate with other externally managed iTracker instances. Therefore multiple iTrackers from different ISPs can optimize their traffic flows jointly. The P4P authors consider the Oracle as a subset of their broader and more generic solution.

5.12.3. Application-Layer Traffic Optimization (ALTO)

Both approaches of Internet service provider assisted optimizations, are currently under review by the Internet Engineering Task Force (IETF). Since 2008 a task force is working on the standardization of traffic localization techniques as the ALTO system [SKS09; SNS+10].

5.13. Network Coordinates based on Matrix Factorization

Another method to predict network distances are matrix factorization algorithms, also called *dot-product* based NC system. The key idea is to represent the $N \times N$ *distance matrix* L using a combination of smaller matrices $L = XY^T$. This technique is based on linear dependencies in parts of the matrix L. These dependencies exists because nodes i, j from

the same AS will most likely share RTTs to other nodes. Therefore rows L_i and L_j will be linear dependent from each other. With this property at hand the $rank(L)$ function is much smaller and the matrix can be expressed as a product of smaller matrices. All dot based NC system assign two coordinates to a node. For each node i their incoming and outgoing vectors $\vec{X}_i, \vec{Y}_i^T \in X, Y^T$ represent a coordinate pair. This reduces the computation of the distance between i and j to

$$d(i,j) = \vec{X}_i \vec{Y}_j = \sum_{k=1}^{d} \vec{X}_{ik} \cdot \vec{Y}_{jk}$$

The advantages of the dot based approach are twofold compared to Euclidean approaches. A separation into incoming and outgoing vectors allows the explicit representation of asymmetric links. An Euclidean coordinate represents only one point in the virtual space. Furthermore, the system can handle violations of the triangle inequality because of the absence of Euclidean geometry's constraints.

Currently there are a couple of dot based NC systems. The first practical solution was IDES [MSS06]. To compute X, Y^T the system uses a fixed set of m landmark server. Ordinary hosts compute these vectors as well when joining the system. In IDES they need to measure mutual RTT distances D^{out} and D^{in} to each landmark server and retrieve each landmark's outgoing vector $\vec{Y}^T$. Based on $\vec{Y}^T$ they compute their own vector pair. As an optimization, ordinary hosts can measure to a subset of the landmark servers and draw information from their vectors. The overall results of the system are close to the results from Euclidean solutions. However, they are not able to match their quality completely.

In [CWS+09] Chen et al. proposed Phoenix, an optimization of IDES. The operation of IDES was relaxed with respect to the information and landmarks servers. In Phoenix, a set of k nodes, called the *early nodes* substitute the landmark servers. To avoid local solutions k needs to be larger that the coordinates dimension d. Each node joining a system with $n < k$ nodes is elected an early node. These nodes need carry out the IDES algorithm to compute their set of vectors and substitute the information server. Normal nodes will use them instead of IDES fixed infrastructure. The second optimization is a concept already used in Vivaldi network coordinate systems, a measure of a node's local error. In simulation based studies the authors claim to match performance of Dabek's Vivaldi.

6. The Hierarchical Vivaldi Algorithm

This chapter introduces a novel concept of using a hierarchy of subspaces to cope with the unavoidable embedding errors, such as the principal inability of the metric space to reflect triangle inequality violations. A key observation of the Hierarchical Vivaldi algorithm is that the embedding errors between a node and each of its neighbors provide additional information that is itself embeddable: In a similar way that latencies define the position of a node in a coordinate system, which we use to predict distances, embedding errors define a position in a "displacement space", which can be used to adapt a node's latency prediction for additional accuracy. The following chapter shows how to construct a hierarchy that embeds both the measured latencies and the respective displacements. Based on such a hierarchy, I propose an algorithm based on Vivaldi, to apply this technique to detect the embedding quality of a given network coordinate.

6.1. Overview

A pure Euclidean space cannot map data that violates the triangle inequality as Section 5.5 of this thesis shows. I already illustrated the negative impact of TIVs on the performance of network coordinate systems. Increasing the dimensionality of the coordinate space does not improve the embedding quality. Using simulations of several Vivaldi variants with up to 32 dimensions in Section 7.4, I demonstrate that the quality of the embedding even degrades when using more than 10 dimensions. However, actual data collected from the Internet shows that violations of the triangle inequality are very common. 70 percent of all data sets in Figure 5.4 are flawed with single hop TIVs. The same applies for the multi hop analysis. Especially the data set provided by the Meridian authors is affected by one hop violations. The King-Blog data is notably flawed with multi hop TIVs. Hence, all Vivaldi variants that have been proposed so far face a fundamental problem.

To overcome this problem, I evaluated other non-metric spaces and found a *pseudo*-Euclidean space as the most promising candidate. That space is both robust and compatible to the concept of Vivaldi. Let me first define the pseudo-Euclidean space [Lau04]:

Definition 1. *A pseudo-Euclidean space E is a real linear vector space equipped with a non-degenerate, indefinite, symmetric bilinear function $\langle ., . \rangle$, called inner product. A pseudo-Euclidean space can be interpreted as composed from two Euclidean subspaces, i.e. E_+ of dimensionality p and E_- of dimensionality q such that $E = E_+ \oplus E_-$. The inner product is positive definite on E_+ and negative definite on E_-. E is characterized by the signature (p, q).*

The inner product between two vectors x and y reads:

$$\langle ., . \rangle = \sum_{i=1}^{p} x_i y_i - \sum_{i=p+1}^{p+q} x_i y_i$$

For $q = 0$ that space is the normal Euclidean space.

However, instead of applying yet another metric, the algorithm uses only one facet of pseudo-Euclidean spaces: Multiple spaces. In Hierarchical Vivaldi I use multiple spaces, to reflect the misplacements of a coordinate, in contrast to a perfect embedding. In particular, I split a pseudo-Euclidean space into two parts, one with positive and one with negative definite metric. The full space embeds the measured RTTs of all nodes. Each part recursively embeds a subset of these nodes. Applying this procedure recursively leads to a perfect embedding of each node, because each displacement is repeatedly embedded, hence adjusted, in the subspace hierarchy. By combining all subspaces I obtain an interval for each predicted pairwise RTT. The spread of that interval corresponds to the prediction quality for the respective node pair.

In the following sections, I formalize that algorithm.

6.2. Embedding Process

For the sake of clarity I start with an embedding of pairwise latency matrix data into an Euclidean space E_0. The embedding of a single measurement l_{ij} updating a node's coordinate is analogous, and is explained after the principal embedding process is clear. Let $L \in \mathbb{R}^{n \times n}$ be the matrix of the measured latencies between the nodes. Let $D_0 \in \mathbb{R}^{n \times n}$ be the matrix containing the distances between the nodes, as computed by the Vivaldi embedding algorithm. Let $\Delta_0 \in \mathbb{R}^{n \times n}$ be the error matrix as defined by the following equation:

$$L = D_0 + \Delta_0$$

In other words L is identical to D_0 after applying corrections in Δ_0. Ideally $\Delta_0 \equiv 0$, which would represent a perfect embedding. The original Vivaldi algorithm cannot yield this result in practice, because e. g. its input data violates the triangle inequality.

Usually Δ_0 is a dense matrix with both positive and negative values because Vivaldi placed some nodes too close, others too far apart. To reflect these different errors, the proposed algorithm splits Δ_0 into two independent matrices, one containing the positive components, the other the negative ones:

$$\Delta_0 = -M_0 + P_0$$

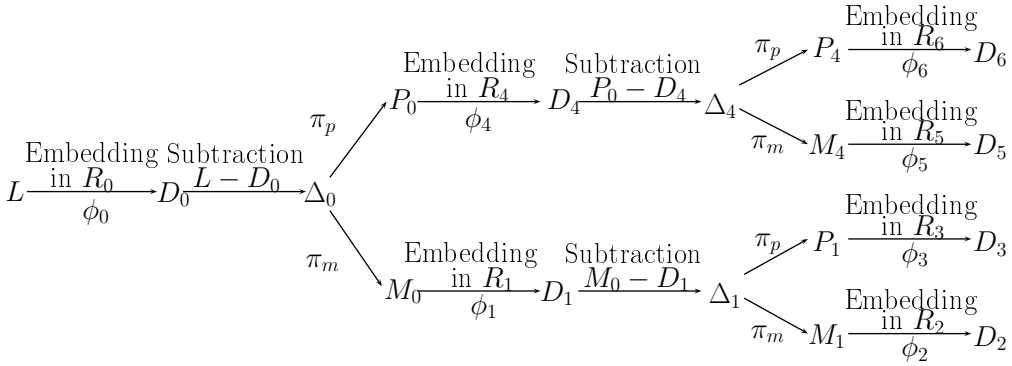

Figure 6.1.: Construction of Hierarchical Embedding $\mathrm{T}_{max} = 2$.

Formally, two functions $\pi_m, \pi_p : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ that operate on a matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ implement that splitting process. I define $m_{ij} \in \pi_m(A)$ and $p_{ij} \in \pi_p(A)$ as

$$m_{ij} := \begin{cases} -a_{ij}, & \text{if } a_{ij} < 0 \\ 0, & \text{else} \end{cases} \qquad p_{ij} := \begin{cases} a_{ij}, & \text{if } a_{ij} > 0 \\ 0, & \text{else.} \end{cases}$$

After the splitting operation, both M_0 and P_0 only contain non-negative components. Now one can apply the Vivaldi algorithm again, to obtain an embedding for the latency deltas in M_0 and P_0. This results in a second order decomposition:

$$\begin{aligned} M_0 &= D_1 + \Delta_1 \\ P_0 &= D_2 + \Delta_2 \end{aligned}$$

Here, D_1 and D_2 contain the distances as predicted by the embedding. Δ_1 and Δ_2 are the respective embedding errors.

$D_{1,2}$ and $\Delta_{1,2}$ are sparser than D_0 and Δ_0, because each node pair is reflected either in D_1 or in D_2, but not in both. Nevertheless, a node is likely to have neighbors such that their RTTs are spread over the $D_{1,2}$ embeddings.

Combining our results so far, we have fractionalized L into:

$$L = D_0 - (D_1 + \Delta_1) + (D_2 + \Delta_2)$$

Every Δ_n matrix can be decomposed further, until the desired depth of recursion is reached. The recursion ends naturally when a D_i embeds only so few latency values that the embedding becomes perfect.

The following definition summarizes this idea and provides the sketch for an algorithm:

Definition 2. *Let $L \in \mathbb{R}^{n\times n}$ be a matrix of pairwise latency data. Let $T_{max} \in \mathbb{N}$ be the maximum depth of the hierarchy and $\eta = 2^{T_{max}+1} - 1$, its number of elements. Let $\mathcal{B}$ be a*

complete binary tree of depth T_{max} with $i = 0...\eta$ that shall be traversed in pre-order. Let $\wp_0, \wp_1, \ldots, \wp_\eta$ be embeddings with $\wp_i = (\phi_i, R_i, d_i)$ for $0 \leq i < \eta$.

$\wp_0, \wp_1, \ldots, \wp_\eta$ is called hierarchical embedding of L at level T_{max}, if $\wp_i$ meets the following conditions:

1. *Each embedding $\wp_i$ is tied to the ith node in $\mathcal{B}$ and has exactly one left and one right successor, unless it is a leaf node.*

2. *The first embedding (ϕ_0, R_0, d_0) is based on the latency matrix L.*

3. *If $\wp_j$ is the left successor of $\wp_i$, $\wp_j$ is based on $\pi_m(L_i - D_i)$*

4. *If $\wp_j$ is the right successor of $\wp_i$, $\wp_j$ is based on $\pi_p(L_i - D_i)$*

On an actual node that runs the Hierarchical Vivaldi algorithm, a single latency $l_{ij} \in L$ is embedded using that hierarchy. Hence, the unmodified Vivaldi or Pyxida algorithm produces the predicted latency $d_{ij} \in D_0$ and $l_{ij} - d_{ij}$ produces the displacement $\Delta_0^{ij} \in \Delta_0$. Depending on the positiveness or negativeness of Δ_0^{ij} a distance in D_1 or $D_{\eta/2+1}$ is predicted. Algorithm 4 shows this modified Vivaldi algorithm in pseudo code. In contrast to the original Vivaldi algorithm it operates on a set of η-sized vectors for coordinates and parameters. The algorithm initially applies the standard Vivaldi (or Pyxida) algorithm to the current coordinate x_n if the latency l is valid in line 5. After the coordinate update, it descends further into the hierarchy, and calls itself recursively with the coordinate displacement of the current subspace. The algorithm descends only either the positive or negative subspace with that displacement, otherwise it is called with a value of 0, which has no effect.

It is crucial to understand that not all the $\wp_i$ embed all the latency pairs. Each latency pair is embedded only in the spaces along a *path* through the embedding hierarchy. The reason is that by splitting Δ_i into a positive and a negative part, the D_i become more and more sparse. A negative latency delta is considered when embedding M_i, only. It is excluded from the embedding of P_i, and vice versa.

Figure 6.1 illustrates the process: Consider a peer (*local node*) and a set of candidate peers (*remote nodes*), which are a subset of all other peers in the system. Based on its coordinate and the coordinates of the remote nodes, the local node calculates an embedding $\wp_0$. For each remote node, the difference between the measured RTT and the distance derived from this embedding determines the further path along which this remote node's latency is embedded. As a result, in each step down the hierarchy there are less remote nodes to embed, and hence the embedding is more and more able to match the input. Finally, if there were only three remote nodes left, the embedding would be perfect (assuming a 2+1 dimensional space).

6.3. Embedding Error Prediction

The algorithm, as described so far, cannot predict latencies from two random coordinates, because only the actually measured latency between two nodes determines the path through the embedding hierarchy. Without that path, comparing two sets of coordinates is not meaningful. Algorithm 4 illustrates that point. In line 3 the latency between two nodes is assigned to variable l. In the hierarchy's lowest order subspace $\wp_0$ l is the actual latency between i and j. Otherwise it is the displacement, which is embedded instead of the latency in spaces $\wp_n$ with $n > 0$. Based on the displacement between l and d_t the algorithm either calls itself recursively with the overestimation m in line 12 or with the positive under estimation p in line 16. Hence, the round trip time determines the path of the algorithm. Therefore, it seems like the algorithm can only be applied to peers that already know their mutual latencies. Nevertheless, the embedding hierarchy helps all peers to optimize their peer selection process. The algorithm idea is as follows.

Every time two nodes are in direct contact, they measure their RTT. With this measured value and their coordinates they are able to perform the described hierarchical embedding algorithm. Thereby, both nodes adjust $T_{max} + 1$ of their η embedding vectors using the Vivaldi algorithm.

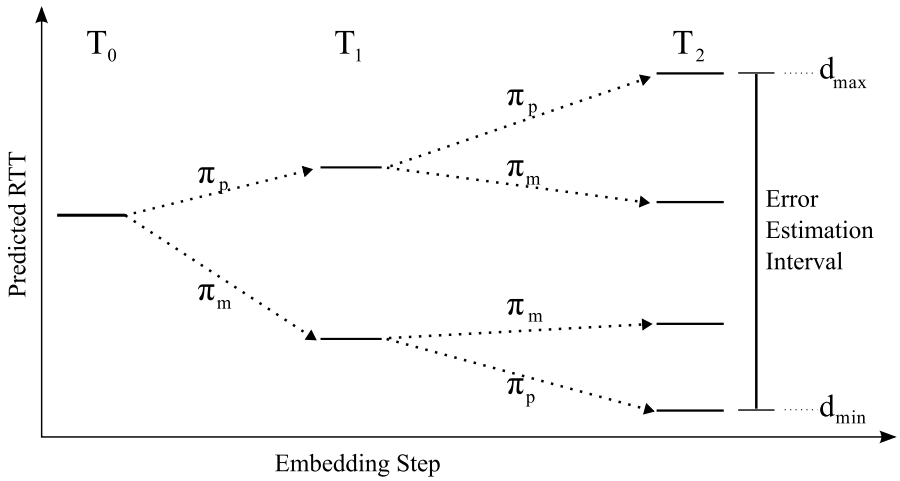

Figure 6.2.: Illustration of the error estimation process.

Assume now two nodes A and B that did not yet have direct contact. They cannot use the embedding hierarchy to compute their predicted mutual latency, because such a calculation is performed along a path in the embedding hierarchy, and determining that path requires an actual latency measurement. Both nodes can, however, calculate the latencies for all the $2^{T_{max}}$ possible paths (cf. Figure 6.2). Only one of these calculated latencies is correct in the sense that it is the value that the hierarchical embedding would produce if it had been given an actual latency measurement between A and B. Hence, the interval, of all possible predictions in an important characteristic:

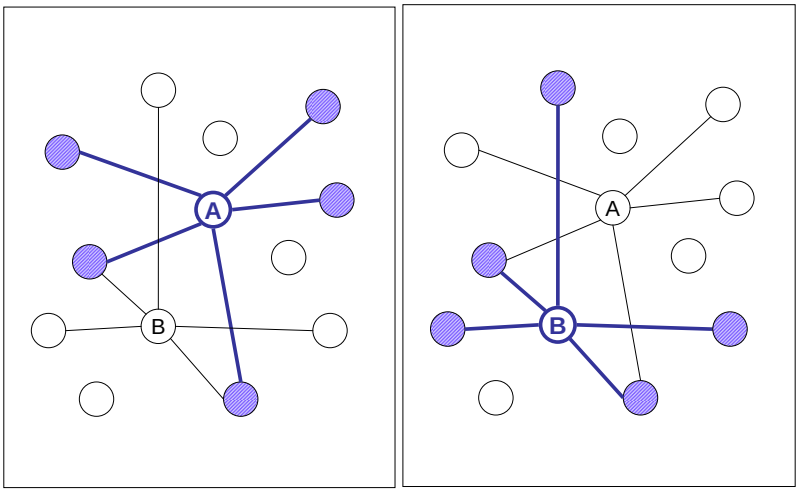

Figure 6.3.: The Peer selection process. Left A's well determined, Right: B's well determined peers.

If the original Vivaldi embedding $\wp_0$ in R_0 is good, the error interval is small and the peer selection process can trust the prediction. Otherwise, the respective peer might only erroneously appear to be a good candidate. Thus, the hierarchical embedding algorithm can predict an interval for the latency. Thereby, it can rate Vivaldi's latency predictions by a margin of error.

Algorithm 5 describes the computation of the margin of error in pseudo-code. The resulting interval $[d_{min}, d_{max}]$ gives us a simple quality measure. The *error estimator* e_e is the relative size of the error interval:

$$e_e = \begin{cases} (d_{max} - d_{min})/d_0 & \text{if } d_0 \neq 0 \\ 1 & \text{otherwise} \end{cases}$$

where d_0 is the latency predicted by 0^{th} embedding.

6.4. Peer Selection Process

The peer selection process can use the error estimator e_e to classify a peer's neighbor candidate. When a peer needs to establish a new overlay connection, it considers preferably its neighbor candidates with a low e_e value only.

It is important to understand that this classification is local to each individual peer. Not all nodes in the network classify their potential neighbors equally. Thus, it is very unlikely that a peer is despised by all other peers. Figure 6.3 illustrates this effect. Node A classifies different nodes as well determined than node B.

Algorithm 4: Hierarchical Vivaldi Algorithm.

Input:
l_{ij}: measured RTT between node v_i and v_j
$\vec{X} = \{\vec{x}_0, \ldots, \vec{x}_{\eta-1}\}$: Local coordinate vector
$\vec{Y} = \{\vec{y}_0, \ldots, \vec{y}_{\eta-1}\}$: Remote coordinate vector
$\vec{\Sigma}, \vec{Z}$: local, remote error vector
$\vec{P}, \vec{\Theta}$: constant vector for c_c, c_e
n: current embedding space number
l: to be embedded rest latency in R_n
t: distance to the trees root
d_t: estimated latency at depth t
m, p: entry in M, P
Constants:
T_{max}: max depth of embedding
Output:
Updated X, Σ

```
1  function hierarchicalVivaldi(l_ij, t, n)
2  begin
3      l ← l_ij
4      if l > 0 then
5          (x⃗_n, σ_n) ← vivaldi(l, x⃗_n, y⃗_n, σ_n, ζ_n, θ_n, ρ_n)
6      t ← t + 1
7      if t ≤ T_max then
8          d_t ← ‖x⃗_n − y⃗_n‖
9          m ← −(l − d_t)
10         if m < 0 then
11             m ← 0
12         n ← hierarchicalVivaldi(m, t, n + 1)
13         p ← (l − d_t)
14         if p < 0 then
15             p ← 0
16         n ← hierarchicalVivaldi(p, t, n + 1)
17     return n
18 end
```

Algorithm 5: Computation of the error window.

Input:
T_{max}: max depth of embedding
$X = \{\vec{x}_0, \ldots, \vec{x}_{\eta-1}\}$: Local coordinate vector
$Y = \{\vec{y}_0, \ldots, \vec{y}_{\eta-1}\}$: Remote coordinate vector

Constants:
T_{max}: max depth of embedding

Output:
d_{min}, d_{max}: RTTs lower, upper bound

Variables:
n: current embedding space number
t: distance to the trees root
m, p: entry in M, P
d_n: distance in R_n
d_u: distance in R_n's parent space
$flpSgn$: flip signedness

```
1  function getMinMaxDistance(t, d_u, flpSgn)
2  begin
3      t ← t + 1
4      if t ≤ T_max then
5          n ← n + 1
6          d_n ← ‖x_n − y_n‖
7          if flpSgn then
8              getMinMaxDistance(t, d_u + d_n, false)
9          else
10             getMinMaxDistance(t, d_u − d_n, true)
11         n ← n + 1
12         d_n ← ‖x_n − y_n‖
13         if flpSgn then
14             getMinMaxDistance(t, d_u − d_n, true)
15         else
16             getMinMaxDistance(t, d_u + d_n, false)
17     else
18         d_min ← min{d_u, d_min}
19         d_max ← max{d_u, d_max}
20     return (d_max − d_min)
21 end
22 e_w = getMinMaxDistance(0, ‖x_0 − y_0‖, false)
```

7. Vivaldi Simulations

The previous chapters introduced the necessity of proximity optimizations, their applications and the proposed solutions. I proposed the Hierarchical Vivaldi algorithm, which promises to be an improvement over existing solutions. This chapter presents the results from simulations of both Vivaldi and Pyxida, which serve as the basis of my thesis. Furthermore, it evaluates my proposed Hierarchical Vivaldi, the major contribution of my thesis in simulations.

I created a packet level simulator that uses latency matrices to create an overlay between peers running the Vivaldi algorithm. The simulator's topology information is based upon several latency matrices from other research institutions and captured latency data recorded during my own studies on PlanetLab. Using these simulations, I developed a sound understanding of the functionality of Vivaldi.

Initially, I investigated the tuning of the Vivaldi variants. Both Pyxida and the original Vivaldi algorithm offer several adjustable parameters. Their complex interaction offers various possibilities to fine tune the algorithm. Based on that analysis I proposed parameter choices in [EFF09] that can improve the algorithms' performance significantly.

Furthermore, I simulated the impact of the neighborhood on the peers' ability to gain an accurate embedding. I experimented with random, optimized and Meridian style neighborhoods.

In the last section I simulated the Hierarchical Vivaldi algorithm. I found that the algorithm reliably detects erroneous coordinates, which leads to a significantly improved peer selection process and much better prediction quality. The simulated environment allowed me to compare the resulting overlay topology to the achievable global optimum. The algorithm misses that optimum only by a few percent. So, it outperforms the previous Vivaldi algorithms by almost an order of magnitude.

7.1. Evaluation Methodology

In order to judge the effect of the existing and proposed Vivaldi algorithms quantitatively, I use a number of indicators throughout the following chapters. To keep the results in the context of existing publications, four of the five measures are from network coordinate literature [PLS05; LGS07; DCKM04]. Only the fifth measure, candidate rank is a variant of its direct predecessor, the RALP value, which captures RALP's information in an improved visual fashion.

1. Relative embedding error The *relative embedding error* [LGS07] between a node i and a node j is defined as the quotient

$$REE_{i,j} = \frac{|\|x_i - x_j\| - l_{ij}|}{l_{ij}} \tag{7.1}$$

where l_{ij} is the actual RTT between both nodes and $\|x_i - x_j\|$ is their estimated RTT based on the coordinates $x_{i,j}$.

Furthermore, [LGS07] defines the relative embedding error of the entire system as the quotient

$$REE = \frac{1}{n^2} \sum_{i,j \leq n} \frac{|\|x_i - x_j\| - l_{ij}|}{l_{ij}} \tag{7.2}$$

This indicator is sensitive to large deviations. Small values indicate only few high-error prone nodes. The n^2 part adjusts the result, as every node is counted multiple times.

2. Median error of an embedding The error of a node pair (v_i, v_j) is the absolute displacement between the predicted RTT $|x_i - x_j|$ and the actual RTT l_{ij}. The error of a node v_i is the median of all errors of node pairs (v_i, v_j). The error of the system is the median of all node errors in the system [DCKM04].

3. Stability *Stability* [LGS07] is the change of a node's coordinates if the other node's RTT stays constant. Stable coordinates are yet another indicator of the embedding quality. They express in particular the utility of cached coordinates. Unstable coordinates are of limited value to other nodes, as they change fast. Recommendations that are based on highly instable coordinates are to lead to wrong decisions and are thus worthless. As most P2P systems include some kind of gossip protocol to exchange — today increasingly proximity based — recommendations, high instable coordinates will lead to wrong decisions and thus render the proximity features worthless.

$$stability = \frac{\sum_i \Delta x_i}{\Delta t} \tag{7.3}$$

Here Δx_i is the drift of x_i in the time period Δt. In the following simulations, Δt is usually set to 2 seconds for simulations of up to 500 seconds, otherwise it is set to 40 seconds.

4. Relative Application-Level Penalty The *Relative Application-Level Penalty* (RALP) [PLS05] is defined as follows:

$$RALP = \frac{1}{n} \cdot \sum_{i \leq n} \frac{|v_i - p_i|}{p_i}, \tag{7.4}$$

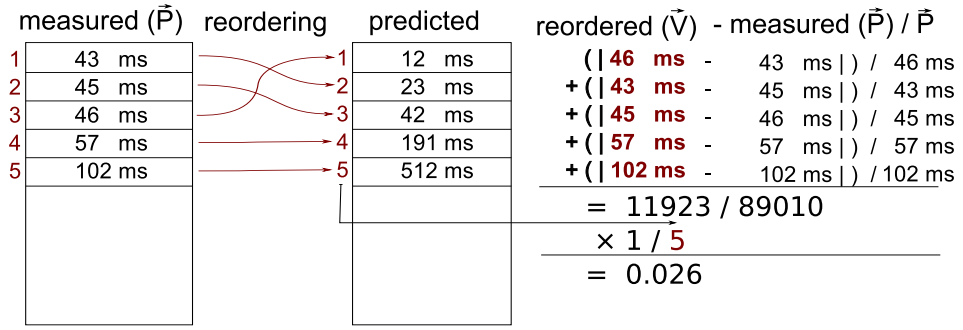

Figure 7.1.: The computations by the **R**elative **A**pp**L**ication **P**enalty measure.

Choffnes et al. [CB09] describe it as the penalty that a node experiences when it chooses a peer based on network coordinates, as compared to the perfect choice that an omniscient oracle could recommend based on actual RTTs. Let $\vec{P} = (p_1, p_2, ..., p_n)$ be a sorted list of RTTs between peer i and n other peers, cf. Figure 7.1. Furthermore, let $\vec{V} = \pi(\vec{P})$ with permutation π be the same list as $\vec{P}$, but sorted based on the *predicted* RTTs between the peer and the recommended peers. In other words $\vec{V}$ contains the *actual RTTs* between the peer and its recommendations, but it is sorted based on the RTT prediction. Figure 7.1 illustrates the different sorting orders. On the left, is the list of measured RTTs $\vec{P}$, in the middle the list of *predicted* RTTs sorted by the *predicted* value. Arrows indicate the RTT's position in that second list. E. g. p_3, which has a value of 46ms is wrongly predicted as a 12 ms distance. The predicted list, resolved to the *actual* RTT ($\vec{V}$) is on the right, written in bold, along with the actual computation. The misalignment of $\vec{P}$ and $\vec{V}$ is what RALP expresses. A value of $RALP = 0$ indicates that the network coordinates predict the next n neighbors perfectly.

RALP captures the impact an algorithm has on the peer selection process. In contrast to other measures it focuses on the actual outcome of a decision. It judges an algorithm's capability to sort nodes based on their predicted RTT. This is also the main focus of any peer selection process, which needs to pick the nearest n peers out of k recommended ones. The relative error of the coordinate is at that point secondary.

A valuable property of RALP is its focus on low latency errors. I computed RALP values from the King-Blog data in Figure 7.2. I started with a perfect prediction vector $\vec{V} : \forall_i : v_i = p_i$, which was increasingly disordered. The x axis indicates how many percent of $\vec{V}$ are disarranged. I took two approaches, represented by a solid and a dashed line in the figure: In the first case, I reordered an increasing part of the low-RTT region of the list: v_1 to v_k with increasing k, in the second case I reordered the high-RTT region from v_n to v_k with decreasing k. Figure 7.2 confirms that the RALP indicator is much more sensitive to errors in the low-RTT region. It responds almost immediately to small errors and is constantly higher, compared to errors in the high-RTT-range.

In the following chapters, the RALP value for a peer i was calculated using $n = 32$ nodes out of 100 randomly chosen nodes that are *not* in the neighborhood of i. Unless

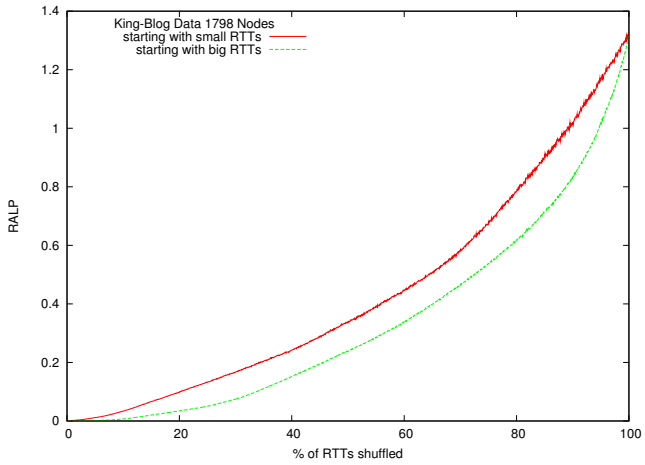

Figure 7.2.: The impact of small and high errors on the RALP indicator.

otherwise noted, all figures show the median of the RALP values for all peers. It is easy to calculate the RALP in the static case because one can extract the RTT values directly from the simulator. In the dynamic case, the latency to all peers needs to be measured, which produces considerable overhead.

5. Candidate Rank Another statistical measure closely related to RALP is the candidate rank (*cr*). It captures the average RTT overhead of a peer when it connects to the k best ranked candidates v_1 to v_k. If a peer fails to find its globally optimal peer p_0 within a reasonable sized recommendation list, the candidate rank expresses its average RTT overhead when it just connects to the e. g. five best ranked candidates.

$$\text{cr} = \{a_0, \cdots, a_k\};\ a_j = \sum_{i=0}^{j} (p_i - v_i)/j \tag{7.5}$$

7.2. Data sets

The simulations in my thesis use both static and dynamic data. Static data is usually described by a latency matrix that represents a snapshot of the measured round trip times or a median of RTTs over a period of time. Dynamic data are the measurements of a fully interconnected mesh of nodes over time. Therefore, the trace file includes real fluctuations in latencies or eventual routing disruptions. Static data sets are smaller than dynamic ones, as they contain only a single latency for a node pair. They are usually captured using a standard method known as the "King" method.

The "King method" is an established procedure, introduced by Gummandi et al. in [GSG02], initially proposed to predict latency between peers. The method is based on the observation that the latency of two peers v_i, v_j is correlated to the latency between

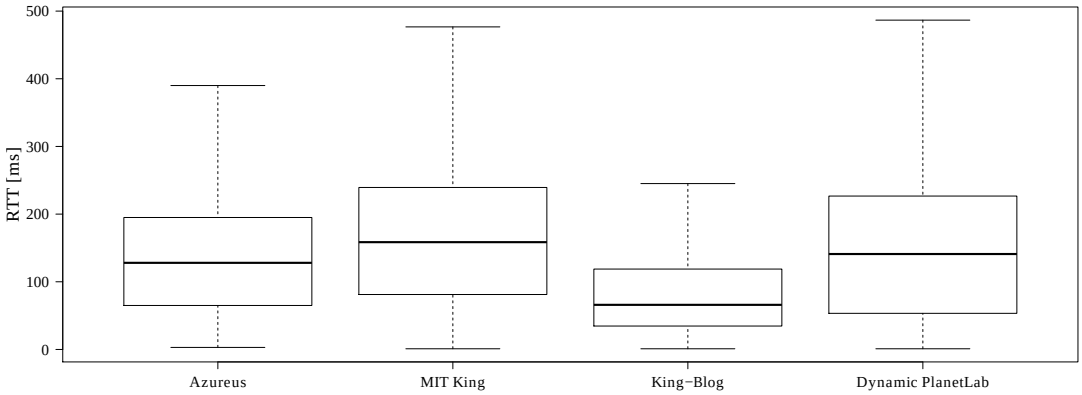

Figure 7.3.: RTT distribution in all data sets.

their closest DNS servers i and j. While King's latency prediction has been superseded by algorithms such as Vivaldi or BBS, the method is capable of collecting latency data between arbitrary DNS servers, a highly popular feature among researchers. King uses recursive queries to deduce these latencies: Initially a query in the authoritative domain of j is issued to node i using a standard DNS resolver from a client node. Unable to give an authoritative answer, i will forward the query to j. Hence, the desired latency l_{ij} is the time between sending and receiving an answer from i minus the latency between the client and i, which is measured in a second step. Of course these measurements are repeated to avoid fluctuations and resolve random subdomains to prevent caching of replies at node i.

The data sets are:

- **Azureus-to-PlanetLab** is the trace from Ledlie et al. [LGS07]. During the development of Pyxida's maturing plug-in, 249 fully meshed PlanetLab nodes supported the existing end user client installations. The trace yields a 249×249 full rank RTT matrix and is available at [AZ].

- **MIT King** is the data set of Dabek et al. used to derive the original Vivaldi algorithm [CDK+03]. It is based on measurements with the King technique among 1740 DNS servers available from [MK0]. Notably, although DNS servers are usually well connected, MIT King is the data set with the highest median RTT cf. Figure 7.3 and variance.

- **King-Blog** is a data set similar to the MIT King data. It was extracted from 2500 DNS servers [Sys06]. With a median RTT of 66 ms it is the data set with the lowest median latency in the simulation cf. Figure 7.3.

- **Dynamic PlanetLab Dataset** is a dynamic trace of 13,4 million single measurements between 83 fully interconnected PlanetLab nodes, which I collected between March 6 and 9, 2009 using the IGOR KBR overlay.

The Dynamic PlanetLab data set, which I captured in 2009 allows a simulation that reflects the dynamics of a real network. For example, Figure 2.3 on page 10 shows a latency collapse from 100 to 300 ms, which was captured in the dynamic trace. In the time from the 6th to 7th. of March 2009. The trace represents a full mesh, whereas subsequent measurements did not reach full rank. Therefore, I focus on that time period. Figure 7.4 analyzes the trace data: For most node pairs at least 2000 measurements are contained in the trace, cf. Figure 7.4 top left. The top right side of the figure shows the number of node pairs with at least one measurement. 6806 measurements per hour are necessary to refresh the whole mesh once per hour. The data set therefore refreshes about 95% of all node pairs in one hour. At the bottom, the figure on the left sums up the number of measurements per node, which is typically between 150.000 to 180.000. Finally the bottom right figure shows the RTT distribution of node pairs.

7.3. Simulator

Chapter 4 introduced the various simulation solutions available in the IGOR ecosystem. In order to evaluate the different Vivaldi variants, IGOR offers ready made and throughly tested implementations. However, to evaluate both static latency matrices and dynamic RTT traces a compiled language is suboptimal compared to comfortable data manipulation functions offered by dynamic languages. Furthermore, the simulation of a TCP stack and the underlying routing using the *OMNeT++* framework is costly and hampers the simulation of huge networks.

Therefore, I built a simulator that is flexible in input formats, offers a simulation queue and re-uses IGOR's Vivaldi implementation. The simulator is able to add jitter to the static latencies in order to simulate small fluctuations common in real networks. Even the simulation of bigger routing disruptions is possible because the user is able to provide an updated latency matrix at any point in the simulation. The method of deriving a Vivaldi simulation from a static matrix of pairwise RTTs was introduced by Cox et al. [CDK+03]. In [EFF09] we extended their design to allow a dynamic trace-based simulation.

On startup, the simulator randomly chooses a neighborhood of 32 peers for each node. If the input data is a static latency matrix, the simulator also determines a sequence of RTT measurements. In the dynamic case, the sequence of measurements is predefined by the trace data. Finally, each node obtains the RTT values for all the peers in its respective neighborhood and the main simulation loop begins.

Simulation Sequence There are two variants of how to determine the sequence of RTT measurements in the static case: In [CDK+03] a node starts a new measurement

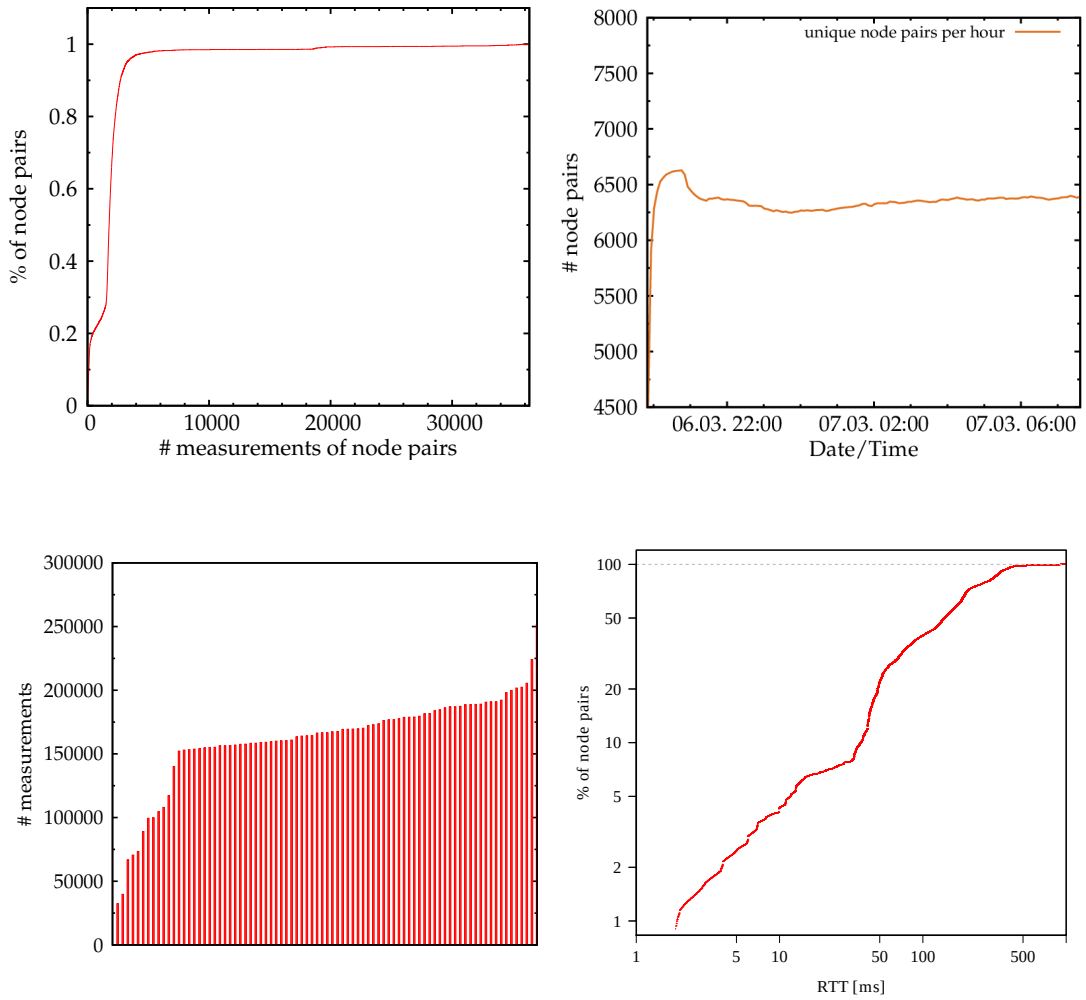

Figure 7.4.: Analysis of the dynamic data trace: Top left: CDF of absolute measurements between node pairs. Top right: Node pairs with measurements within a hour. Bottom left: Absolute number of measurements per node ID. Bottom right: CDF of RTT distribution in the data set.

immediately after the previous measurement has completed (*continuous adjustment*, CA). This leads to an imbalance because nodes with low RTT conduct more rounds of the Vivaldi algorithm. Another drawback is the huge traffic overhead that those continuous measurements cause. In order to avoid this imbalance and reduce the overhead, I compare CA to a variant that uses predefined average time intervals for the measurement rounds (*uniform adjustment*, UA). Each node measures each neighbor only once per round. To allow each node to conduct as many measurement rounds as the node pair with the lowest latency in a 500 seconds CA simulation, UA simulations stop not until 20 000 seconds. A round is ten seconds long. To reflect the different adjustment modes, the Δt parameter of the stability quality measure is also adapted. Continuous adjustment sets the delta to 40 seconds while uniform adjustment sets it to 1000 seconds.

7.4. The impact of Triangle Inequality Violations

I discussed the negative impact of triangle inequality violations both in Chapter 2 and Section 5.5. A Euclidean space cannot map non-Euclidean data. To measure that impact, I enhanced a data set to reflect perfect metric latency data. Preprocessing the Azureus data produced an artificial data set, where all pairwise latencies were raised so that they did not cause a violation of the triangle inequality any more. That data set was chosen for its smaller percentage of TIVs ($\rho < 1$) in Figure 5.4 (black dashed line) and for its size. Resolving all TIVs is a computational hard task, therefore I chose a smaller data set.

Figure 7.5 compares Vivaldi's performance facing a moderate number of TIVs (left part) versus a TIV-stripped version (right part) of the Azureus data set. The top part of the figure shows the chronological sequence, the bottom a CDF view at the end of the simulation. Each figure uses the same Vivaldi configurations. I focussed on Vivaldi embeddings in spaces of different dimensions. Comparing Vivaldi's performance on both the modified and the original data, the figure shows a median embedding error of under 3% compared to 6% for each best performing simulation. The negative influence of TIVs on network coordinate systems therefore is severe, which is also discussed in different publications [LBL+09].

The figure reveals another notable point. The height enhancement of the Euclidean model is able to embed TIVs. Comparing both CDF plots, the best performing configuration on the left is 9D+Height, along with 4D+Height. The right side in contrast exhibits a favor for the conventional Vivaldi variant, after removing all TIVs. This result indicates that the height configuration scored best results in the left case for its better performance facing TIVs. And indeed, Figure 7.6 shows the measure ρ using coordinates, gathered from a Vivaldi embedding with negative heights. Both figures indicate that the resulting embeddings for all static data sets do violate the triangle inequality, which is impossible for the Euclidean embeddings produced by a non-height Vivaldi. An algorithm that is based on an Euclidean space normally produces coordinates for which $\rho > 1$. Hence, using a negative height component mitigates TIVs to a certain amount.

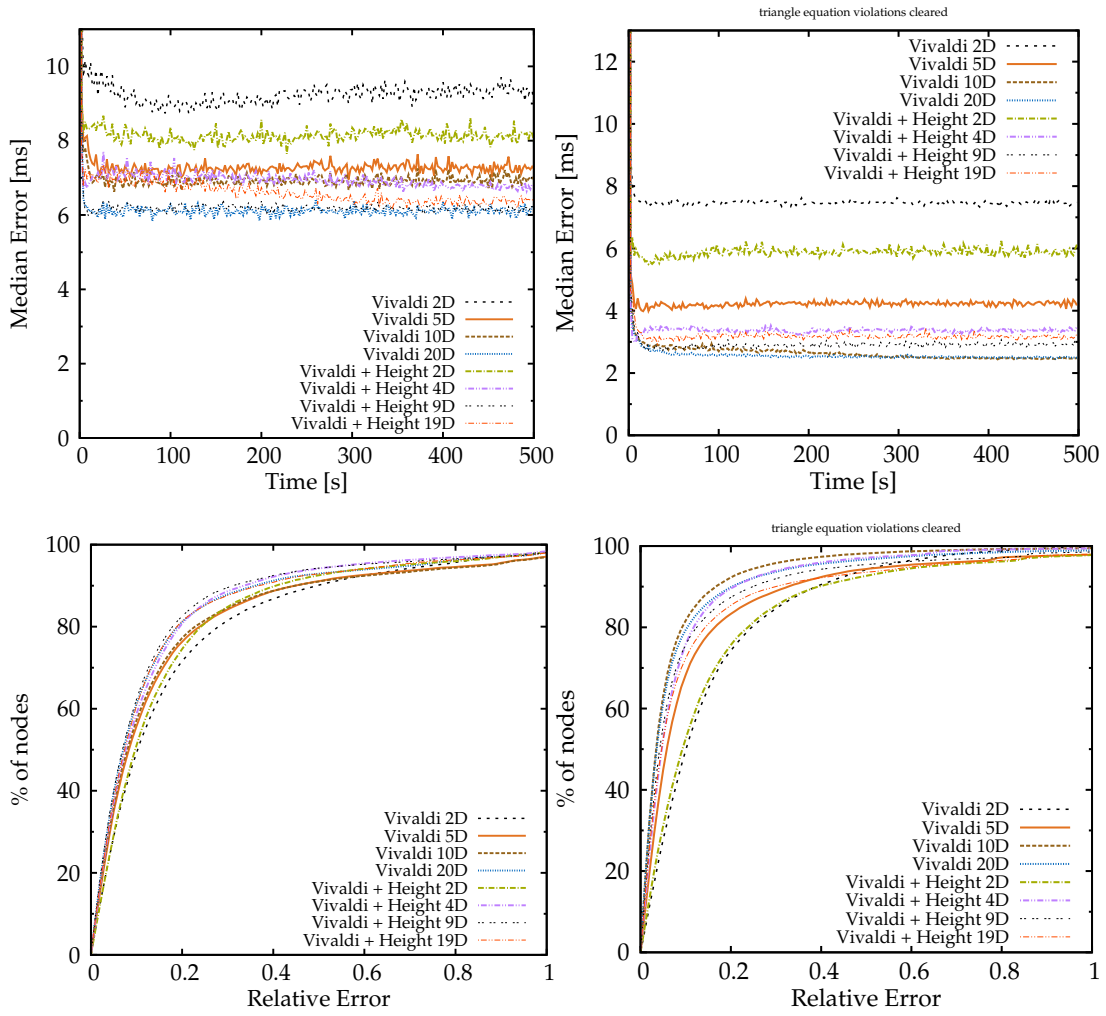

Figure 7.5.: Triangle inequality violations impact on Vivaldi's performance. Left: Azureus data, Right: TIV-cleared Azureus data.

Finally, Figure 7.5 indicates the impact of dimensions. After an increase in performance, the use of high dimensions does not yield better values. The best performing configuration is a 10 dimensional Vivaldi, with or without height depends on the TIVs of the data. Higher dimensions, even with two times higher dimensionality introduce a notable overhead at no measurable benefit.

7.5. The choice of Neighbors

Starting with [CDK+03] PNS and PNR stress the importance of choosing latency close neighbors in order to improve the performance of an overlay. But the opposite side, how

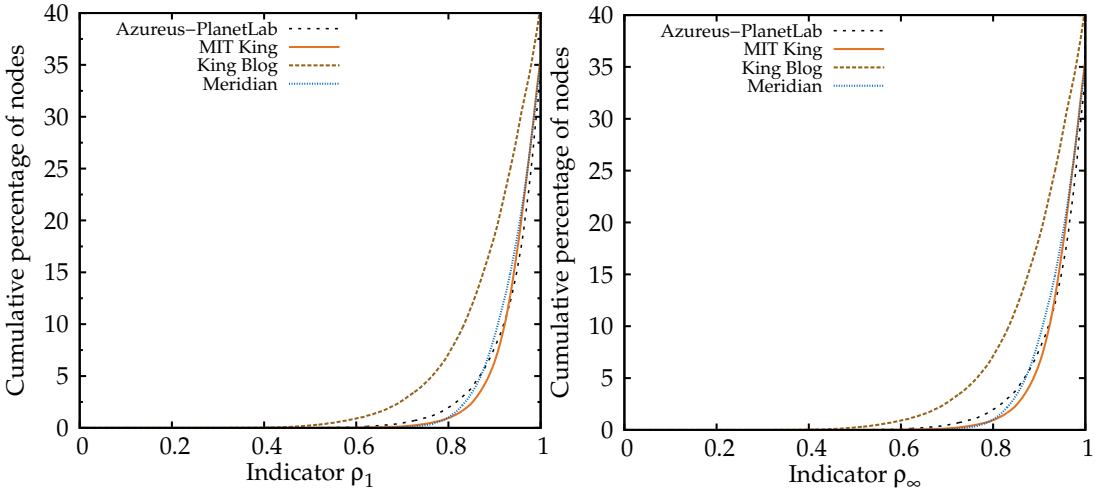

Figure 7.6.: Single-hop (left) and multi-hop (right) violations of the triangle inequality using negative heights.

neighbors in the overlay influence the performance of the Vivaldi algorithm itself has never been studied.

Therefore, I conducted a simulations study, using the simulator, described earlier in this chapter. Allowing each node to take each possible neighbor, maximizes the flexibility of neighbor selection. Therefore, I used a static simulations, as they offer a full rank latency matrix. Because the King-Blog data spans the largest matrix of 2500×2500, I chose the latter for the simulation study. Each node had at least 32 neighbors. I define the following scenarios, to supply each node with different neighbors:

- **Normal:** Neighbors are assigned uniformly random, i. e. with no preference for particular nodes. The neighborhood is static.
- **Normal Optimized:** Similar to Normal, but removes the neighbor that provoked the highest relative error in the most measurements during the Normal simulation.
- **Normal Meridian:** After a standard assignment of neighbors, nodes update neighbors based on recommendations, similar to Meridian closest node discovery.
- **RTT-Close:** Uniformly random selection of neighbors out of the 250 latency closest nodes.
- **RTT-Close enhanced:** Similar to RTT-Close, but with the most erroneous node removed.
- **RTT-Close Meridian:** Starting with a subset of the 250 latency closest nodes, nodes update neighbors based on recommendations.
- **Perfect:** The closest nodes of each peer.

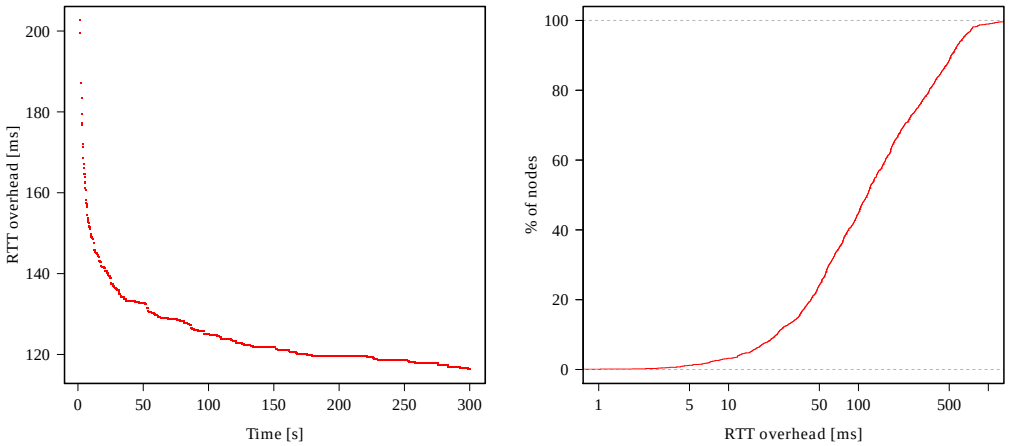

Figure 7.7.: The efficiency of a network coordinate based Meridian algorithm. Left: Median candidate rank computed by comparing the Meridian optimized neighborhoods with an omniscient oracle. Right: Distribution of candidate rank after 300 minutes.

Proximity Neighbor Selection: The simulator implements a PNS algorithm that is based on the design of the Meridian algorithm. Upon contact from node A to node B, the receiving node B selects a set R from its neighbors. Elements in R are chosen based on their Vivaldi distance to A. For the simulation study $|R|$ consists of a maximum of 4 elements. Upon receiving R, node A assures the correctness of the information using direct latency measurements. If a recommended node is closer to A, than an existing neighbor that existing node is removed. The algorithm resembles Meridian's gossipping protocol, by spreading closest node information among close peers. It replaces Meridian's need for direct latency measurements by a coordinate based prediction. Figure 7.7 illustrates the achieved accuracy of the PNS algorithm. Starting with a total random neighborhood, I compute the candidate rank each node by comparing a node's current neighborhood with an omniscient oracle. The left figure, illustrates that after a short initial period, the median overhead is cut to 130 ms and finally reaches reaches a 50 % overhead cut off below 120 ms. The cumulative distribution on the right shows the situation at the end of the simulation: More than 200 nodes include their 32 closest neighbors within a 20 ms overhead cf. right figure. The algorithm is stateless. A node will always recommend the $|R|$ closest peers, therefore it is more likely to recommend an already known node to the receiver. Therefore, the change rate (the left part of Figure 7.7) diminishes.

During my simulations I discovered substantial differences between Vivaldi's performance, depending on the choice of neighbor nodes for Vivaldi. In Figure 7.8 a clear favor for a random choice of neighbors emerges. On the left side of the figure the median relative error of a node shows that a static random neighborhood will outperform all other variants. Even the Perfect choice, which could naïvely be considered as the best choice

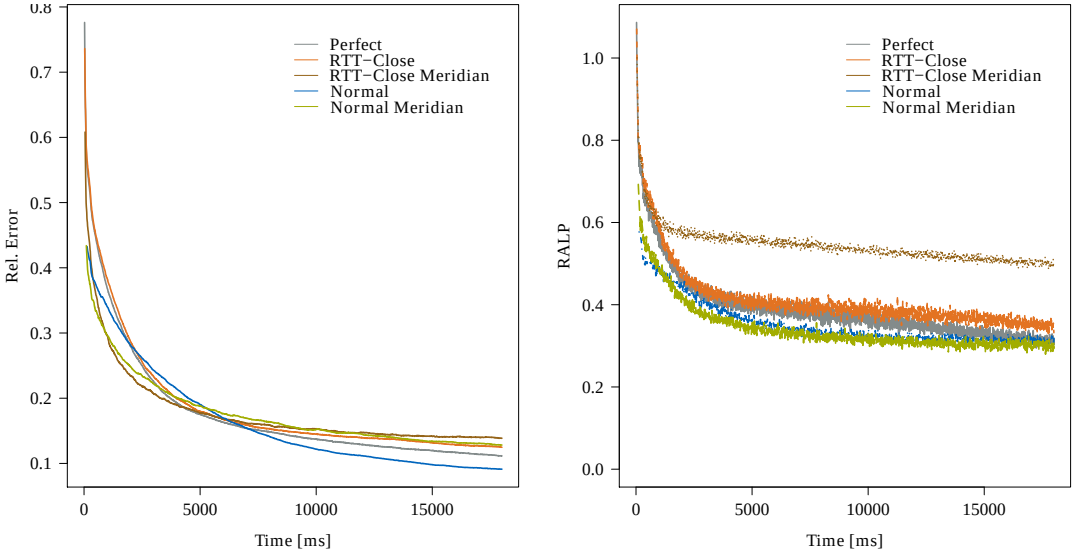

Figure 7.8.: Various Neighbors: Relative Error (left) and RALP (right) comparison (CA).

exhibits only mediocre results. Even after substantial run time, the algorithm shows different behavior comparing the various neighbor set ups. The RALP measures on the right side of the figure testify good performance for the Meridian random variant, but still affirm a preference for random neighborhoods. Both figures affirm that a unilateral choice of neighbors could converge to a suboptimal solution in the long run.

7.6. Vivaldi Simulation Results

In this section, I introduce the results from my extensive study on the existing Vivaldi variants. I analyze all the proposed algorithm variants with the four data sets and with various parameter settings. From that simulation I derive a recommended set of parameters that allow Vivaldi to quickly find its position in the coordinate space and react conservatively to routing disruptions.

7.6.1. Static data

The following section introduces the results from simulations using static latency matrices. These static data sets allow a comparison of the different RTT probing intensities, which I previously introduced: Continuous adjustment and uniform adjustment.

Vivaldi:

Figure 7.9 shows the error, Figure 7.10 the stability of the original Vivaldi algorithm with continuous adjustment (CA). The figures show three static data sets and different values for c_c and c_e. In accordance to Cox et al. [CDK+03] the figures indicate that smaller

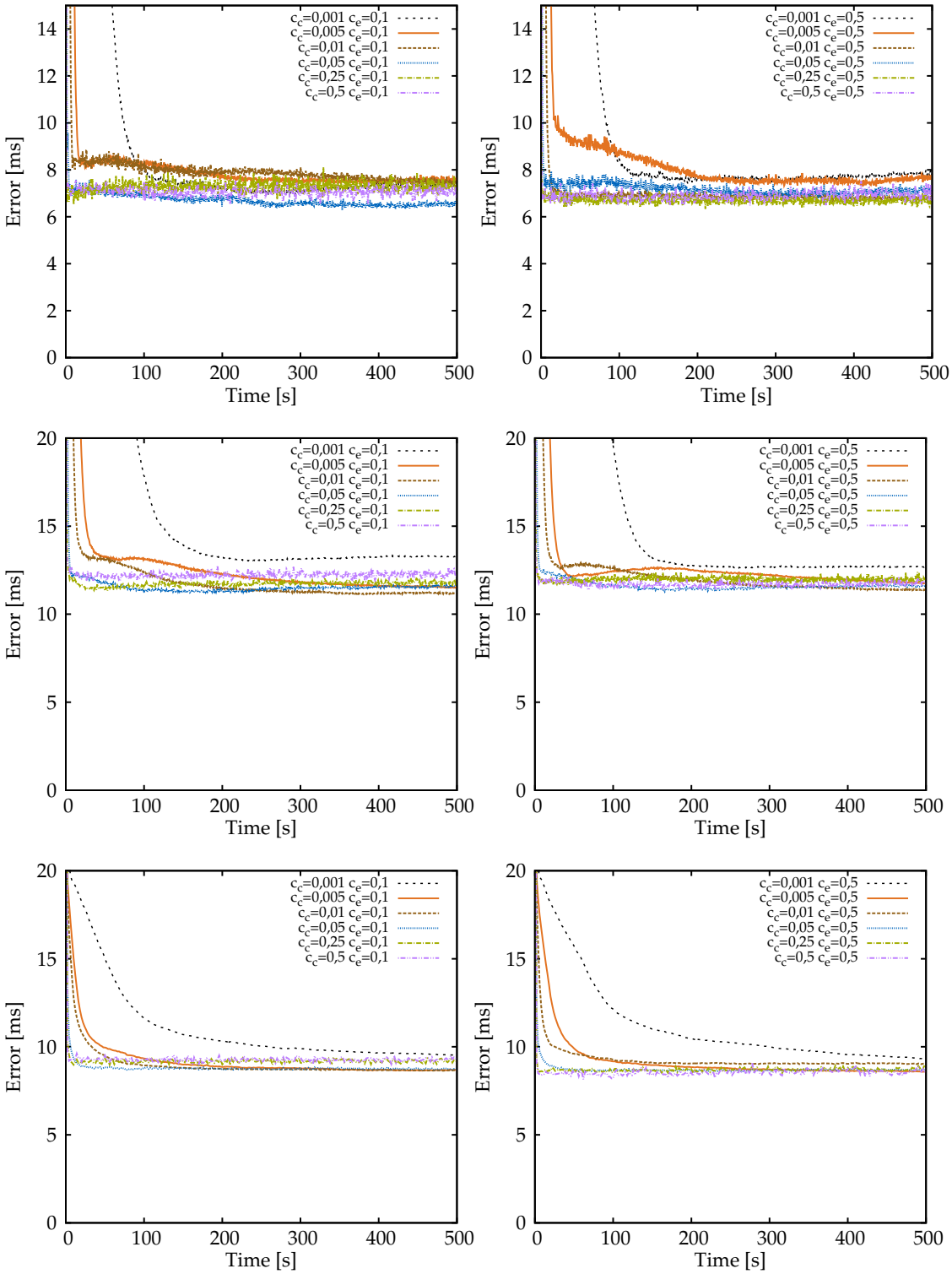

Figure 7.9.: Median error for Vivaldi simulations using contiunous adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

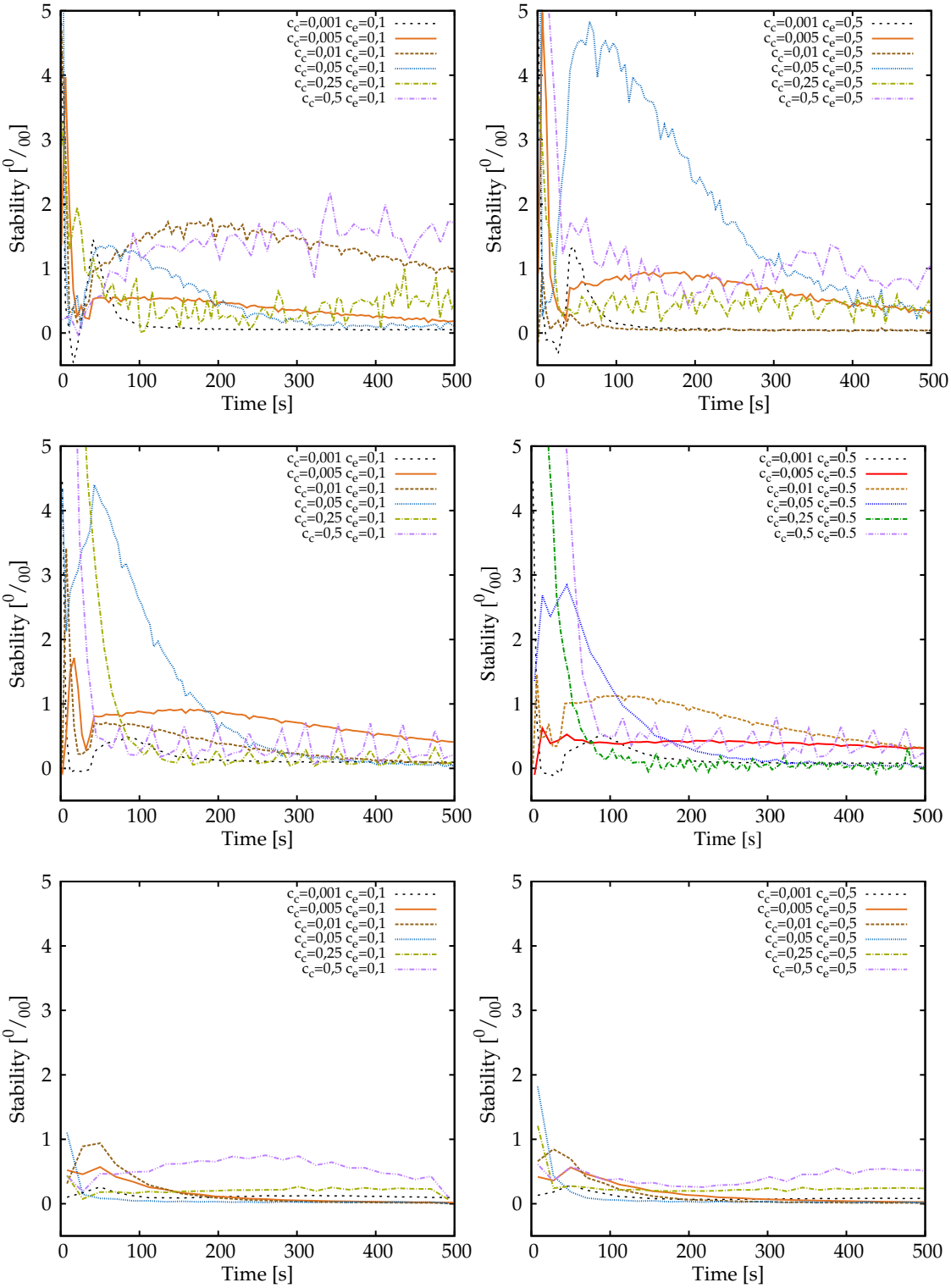

Figure 7.10.: Stability for Vivaldi simulations using contiunous adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

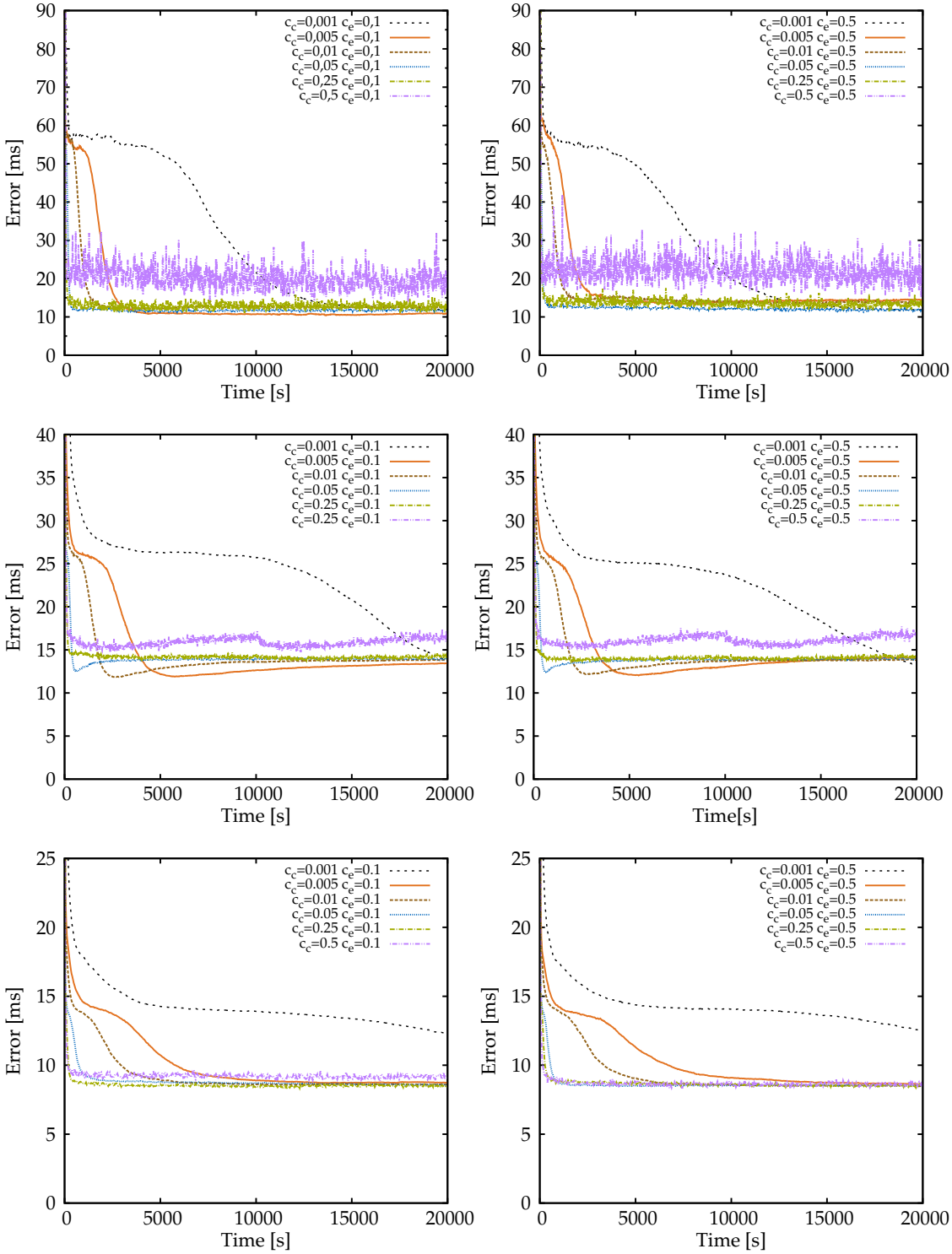

Figure 7.11.: Median error for Vivaldi simulations using uniform adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

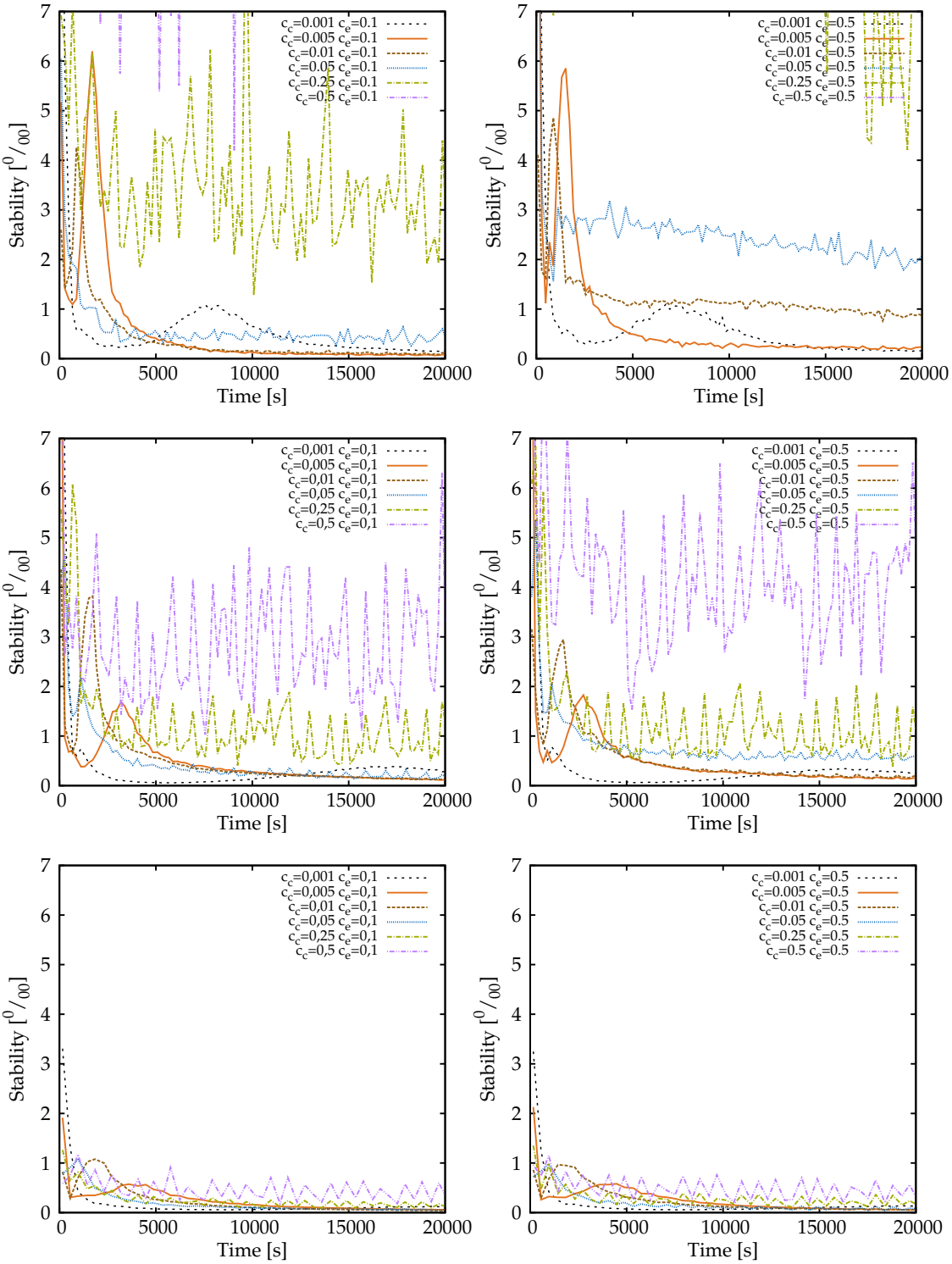

Figure 7.12.: Stability for Vivaldi simulations using uniform adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

values for c_c and c_e lead to more stable coordinates. But as the analysis suggests, this improved stability is only relevant in the initial phase: After 500 seconds, the stability of the embedding is similar for all parameter combinations. Also adjusting parameters has only limited impact on the achieved error. However, the static simulation is designed with static neighborhood and only modest jitter. Therefore, choosing more conservative parameters that quickly adapt and do not exhibit huge instabilities is a sensible choice. Moreover, I found through the course of all simulations that c_e has only little influence on the embedding error and the stability. A conclusion that is also supported by both figures. Finally, the diversity of the data sets becomes apparent: For example, the stability of the upper data set (Azureus) compared to the lower data set (King-Blog) differ in both in the absolute values and in the time when both data sets reach stability. Five out of six simulations in King-Blog reach stability with $c_c = 0.1$ after 100 seconds, but the identical setup takes 200 seconds to reach that point using the Azureus data. However, stability does not automatically include accuracy: The Azureus data, which happens to be quite unstable, reaches the best relative error in the Vivaldi experiments.

Figure 7.11 and Figure 7.12 show the same scenario for the uniform adjustment (UA) variant. Due to the much lower measurement frequency the shown time scale covers a larger range. In spite of the similar number of measurements it stands out that both figures do not reach the accuracy of the CA variant. In all settings, both stability and median error are worse. Previously, the Azureus data set reached a minimum median error of 7 ms. Now the picture is different: The King-Blog data exhibits the lowest error, with 9 ms, Azureus reaches only 10.7 ms. Furthermore, we see that small c_c values delay the convergence enormously, whereas large values lead to great instabilities. For large c_c values the low stability also leads to a high, fluctuating embedding error. As a result I propose an optimal parameter choice at $c_c = 0.005$. The c_e parameter has limited effect, however, a value of 0.5 leads to higher instabilities in combinations with bigger values of c_c. Similar to the CA variant, King-Blog is the most stable data set throughout all parameter settings, and it is remarkably robust to the variation in measurement frequencies. It also exhibits the same relative error, while scarifying only some stability.

Pyxida:

Figure 7.13 and Figure 7.14 show the Pyxida optimization of Vivaldi for continuous adjustment, Figure 7.15 and Figure 7.16 for uniform adjustment. On the left column of each figure, Pyxida's e_t parameter is set to the median latency of each data set. The right column is using $e_t = \infty$. This parameter choice causes the peers to adjust their position relative to all the peers that they ever contacted. That setting reflects the choice of Ledlie et al. in [LGS07]. The same setting is used in the UA variant: While the right column uses ∞, the left uses the median RTT value. The median parameter choice causes the peers to adjust their position only relative to active peers, i.e. those that have just sent their RTT

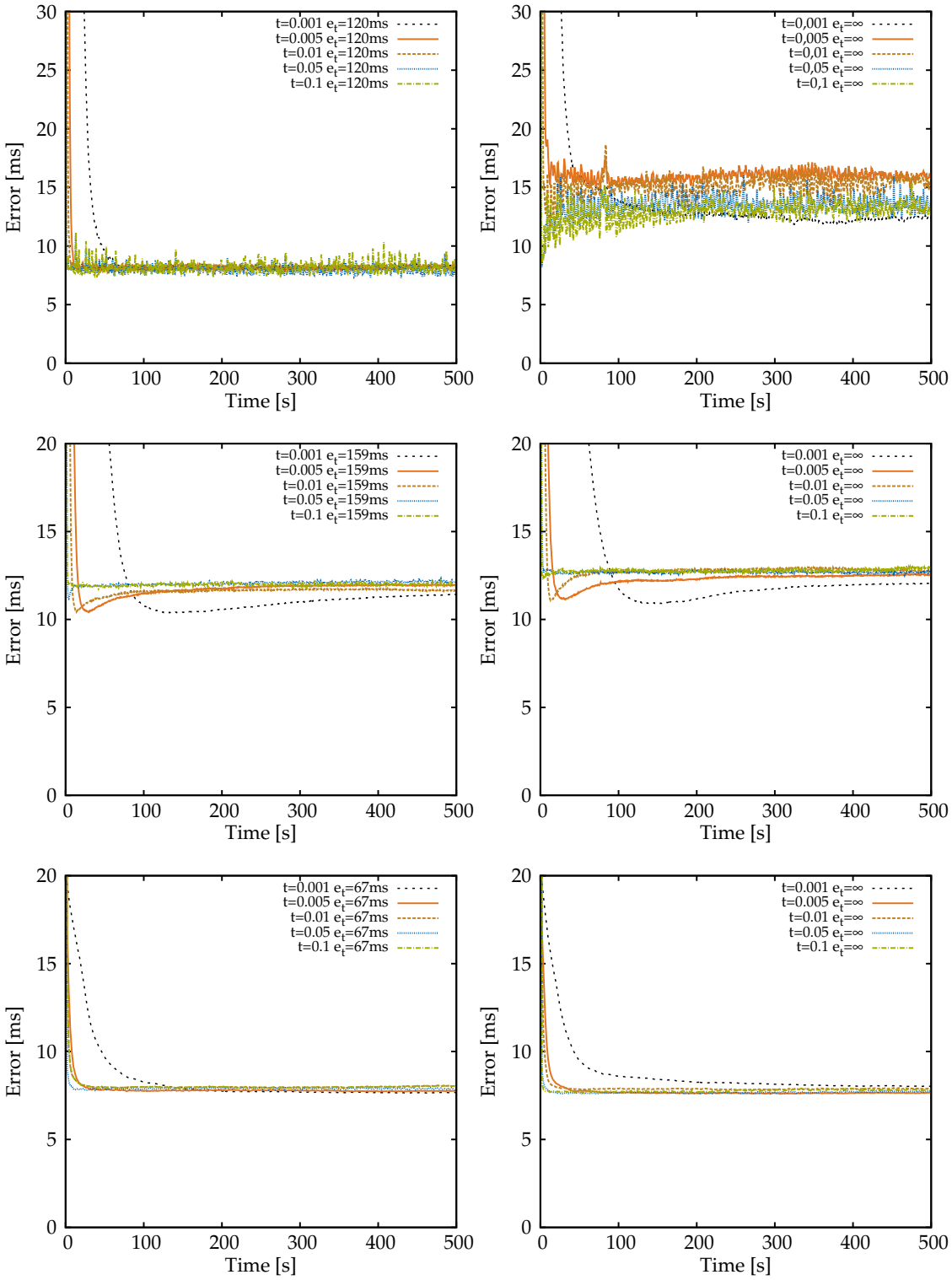

Figure 7.13.: Median error for Pyxida simulations using continous adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

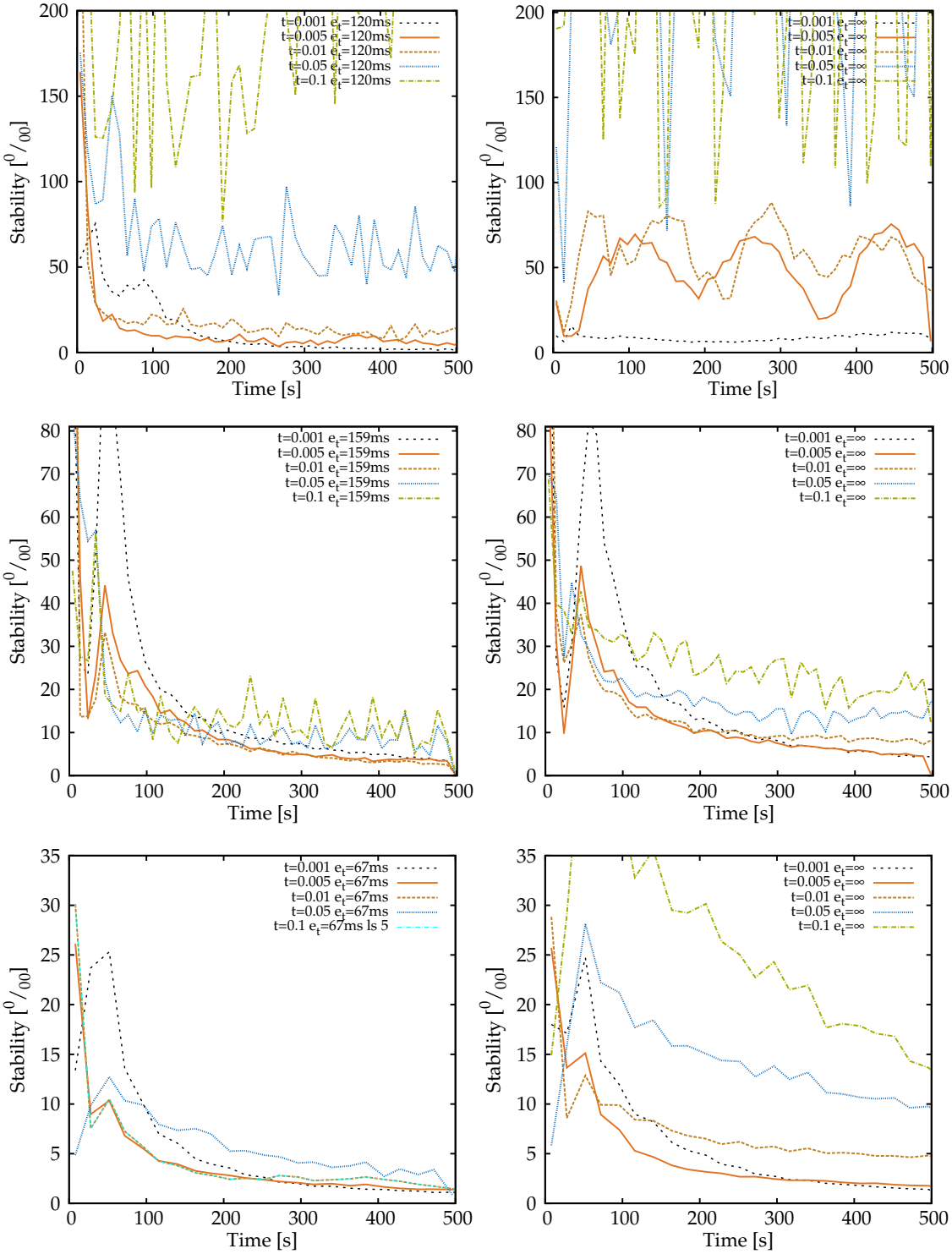

Figure 7.14.: Stability for Pyxida simulations using contiunous adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

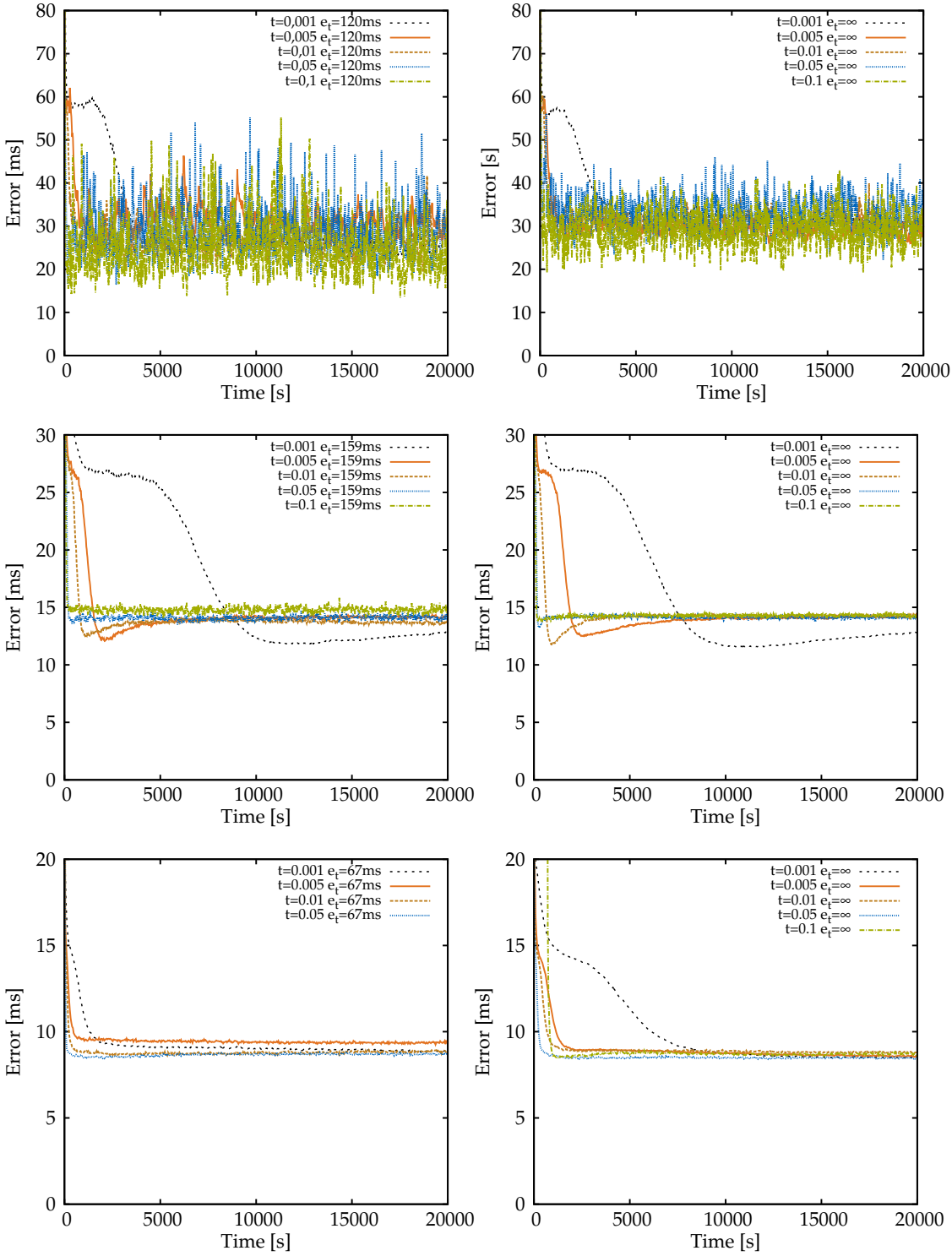

Figure 7.15.: Median error for Pyxida simulations using uniform adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

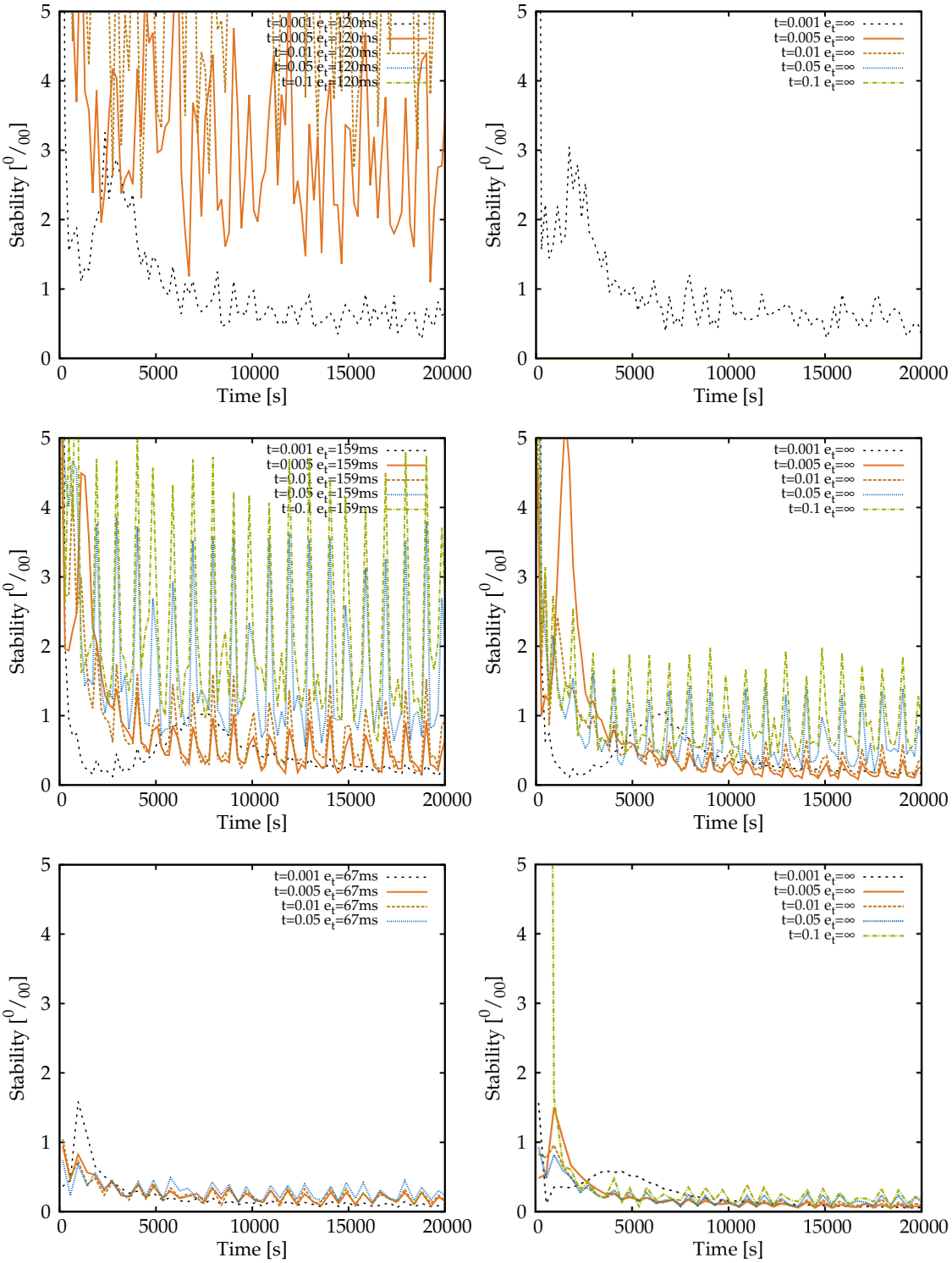

Figure 7.16.: Stability for Pyxida simulations using uniform adaption. Top: Azureus data. Middle: MIT King. Bottom: King-Blog data sets.

information. When using CA a node will update its coordinates with any other node and a set of its recent other nodes, which is a dynamic subset of it's neighborhood. UA in contrast assigns fixed time slices, hence nodes the set of other nodes is a static subset. The difference is marginal, but might well explain the jump in accuracy that happens when comparing the relative error. While King-Blog is robust against that change, it has a modest change from 7 ms to 9 ms, both other data sets experience jumps from 10 to 15 ms (MIT King) and even 10 to 20 ms (Azureus). King-Blog is probably the least affected set for its lowest median RTT and small variance.

As in the Vivaldi simulation, the achievable embedding quality depends on the adjustment variant. Better results are achieved using continuous adjusting, at the cost of a sizable communication overhead. Stability-wise $e_t = \infty$ compared to a median e_t setting provokes the largest difference. Using continuous adjustment one finds significant stability improvements for $e_t = \infty$, but using uniform adjustment the opposite is true. The huge neighbor decay provokes a significant instability.

RALP

In order to complement the analysis with an application-level metric, I further analyzed the different variants using the RALP quality measure. Figure 7.17 shows the results for continuous adjustment (left) and uniform adjustment (right). Clearly, all variants improve over the random peer selection case, which does not use network coordinates at all.

Just like in the previous sections, the difference between the two adjustment variants depends on the data set. Simulations with the KING data set (bottom) behave comparably for all variants (Vivaldi, Pyxida, w/ and w/o CA, UA) of the Vivaldi algorithm, whereas the Azureus data set exhibits a significant dependence on the parameter choices. In particular, when using the uniform adjustment variant, the original Vivaldi algorithm outperforms the Pyxida optimization. Also the King-Blog data shows no significant improvements of Pyxida over Vivaldi. This indicates that the claimed improvement of Pyxida [LGS07] might not carry over to a general application 'in the wild', despite the suggestive title of the respective publication.

Conclusion

The end of the static simulations marks also the end of comparisons for the different different RTT probing intensities. I analyzed all data sets using different measures, to determine the impact of that very change in measurement frequency. The results demonstrate that the UA variant reproduces the CA results within a factor of two, while requiring only sparse RTT probing (10 sec versus 120 ms). Due to the lower measurement frequency the time until coordinates reach stability is more than ten times higher and some accuracy is sacrificed to the lower probing traffic. However,

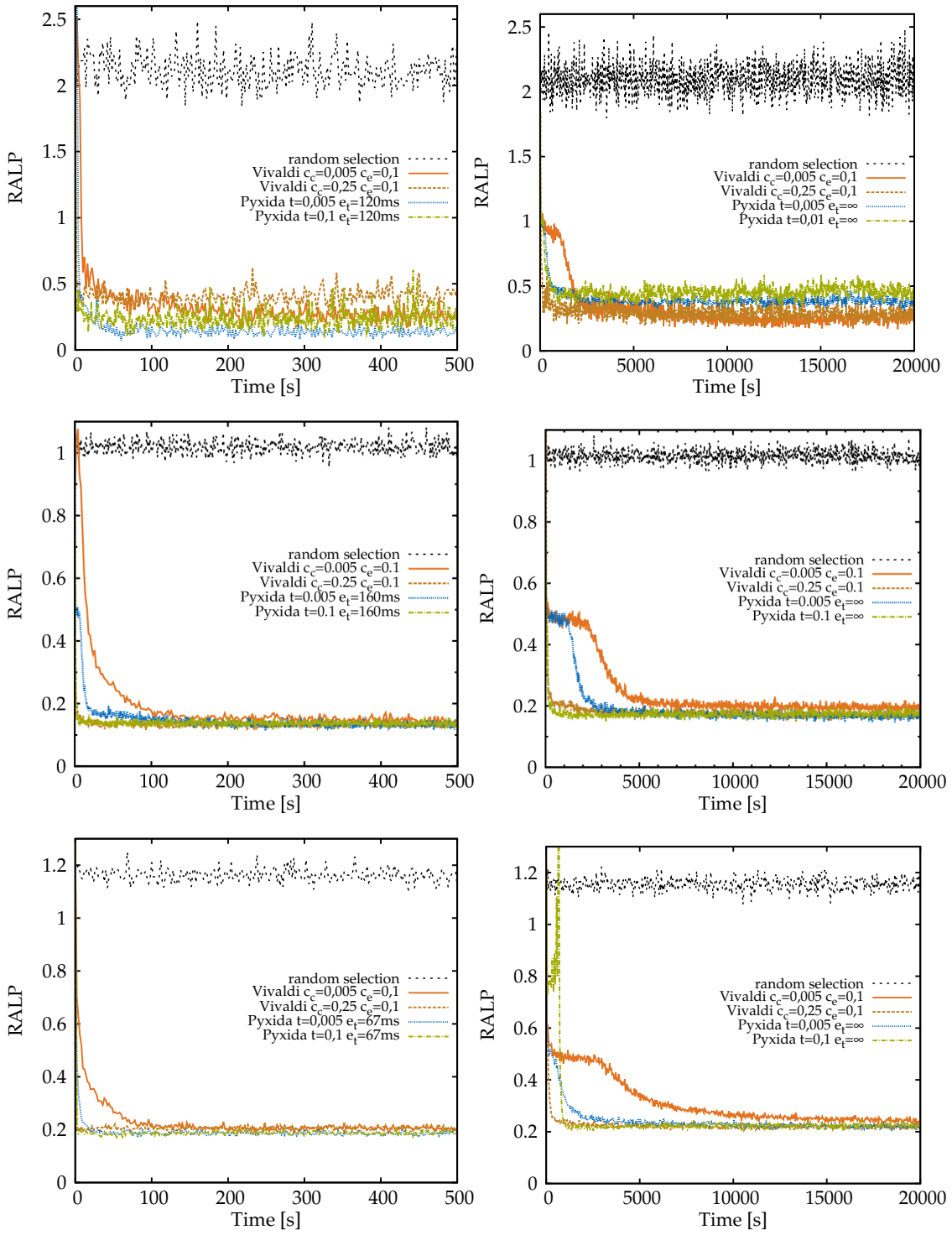

Figure 7.17.: Comparison of Vivaldi and Pyxida using the RALP measure. Left: CA, Right: UA.

both variants represent extreme cases. The first variant always sends data, the other only seldom. These results therefore represent a best and a worst case scenario to the designer of a network coordinate enabled application. Furthermore, the RALP measure, a measure designed to be application specific exhibits only marginal differences between both variants´ end results. Therefore, one can conclude that performance is satisfying in both cases and the results of an actual application are in the illustrated range.

7.6.2. Dynamic data

In order to better understand highly dynamic systems and their differences to static systems, I captured and analyzed the PlanetLab RTT trace. The simulation was set up, in a way that each node was randomly connected to 32 neighbors. Both the measurement sequence and jitter of latencies were determined by the captured data in the trace.

A highlighted feature in many publications about the Pyxida algorithm is the median filter. It was proposed to mitigate fugacious and erroneous changes of latencies. Therefore, I analyzed its influence on the actual results of a simulation. I carried out two identical simulations rounds, with and without that filter. The median filter operates on the last 30 minutes of measurement data, Pyxida's e_t was set to the same value.

Figure 7.18 shows the embedding error for the Vivaldi and Pyxida algorithm using the various parameter combinations from the previous chapter. The top figures shows the results for Vivaldi, the bottom figures the results for Pyxida. All of these simulations were carried out with the median filter proposed by Ledlie et al. In contrast, Figure 7.21 shows the same data and arrangement without this filter enhancement. First of all, there is no significant observable effect of the median filter. Both median errors are in a nearly identical range. Furthermore, the advantage of the Pyxida optimization manifests the most when it is compared to a high c_c parameter in the original Vivaldi algorithm. Figure 7.19 puts a spot on the best performing parameter sets with latency filter (left) and without (right). Clearly, when c_c is set to 0.01 Vivaldi and Pyxida behave almost identically. Furthermore, the latency filter has only a minimal impact on Pyxida's median error, while Vivaldi performs slightly worse.

Figure 7.20 illustrates the complete simulation setup stability wise. Δt was set to a value of 1.5 hours with up to 10 nodes in the set. All measurements show large instabilities for large values of c_c and t respectively. Again, it is apparent that the choice of large values for c_c and t does not only lead to faster convergence (as stated in literature), but also to an increased instability that in the end leads to a worse quality of the embedding. Therefore, it is apparent that even though the Pyxida optimization can yield a better stability in principle, it does only minimally improve the embedding for the moderate values of t and c_c that yield the best results overall.

As a final step, I re-examined the dynamic setting, just like the static data using the RALP measure in Figure 7.23. Here I find that the Pyxida optimization shows a

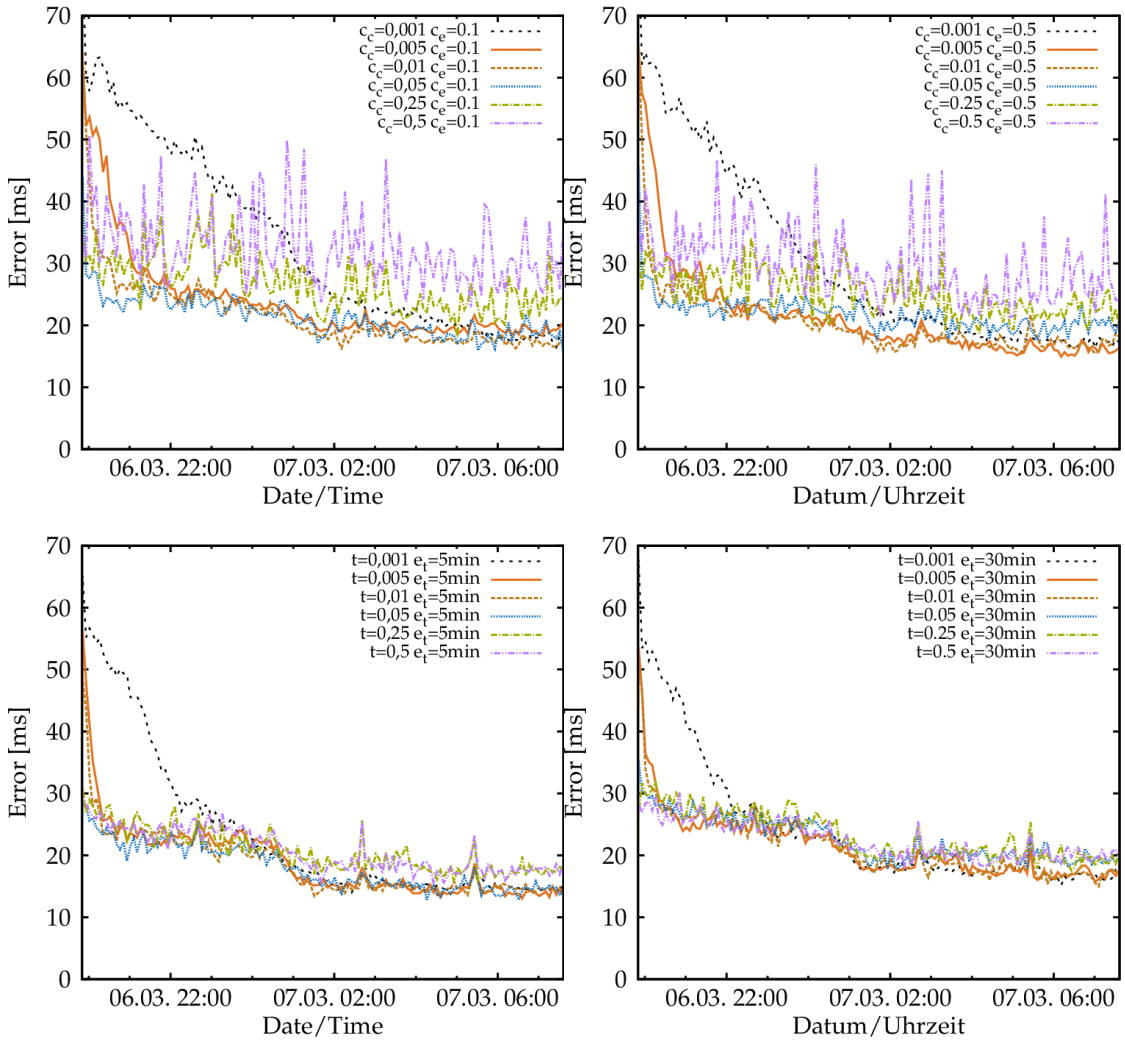

Figure 7.18.: Median error for dynamic trace simulations, using a RTT median filter. Top: Vivaldi algorithm. Bottom: Pyxida algorithm.

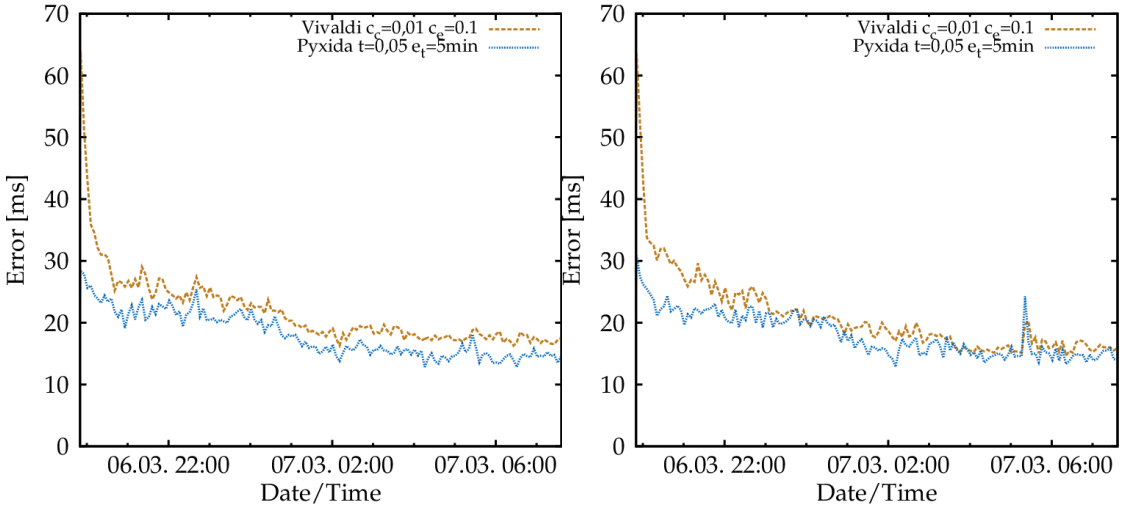

Figure 7.19.: Comparison of the best performing Vivaldi and Pyxida set ups. Left: With median filter. Right: Without.

slightly better performance than the original Vivaldi algorithm. Again, $c_c = 0.005$ and $c_e = 0.1$ outperform all other parameter combinations in the Vivaldi case. For the Pyxida algorithm $t = 0.005$ is the best choice.

7.6.3. Conclusion

I draw several conclusions from the simulations in that chapter:

First, most constants are chosen too high to produce stable coordinates in dynamic real world settings in the original publications. Seemingly the respective authors decided to sacrifice the stability of the resulting embedding to obtain a faster convergence. I believe this to be a bad trade, especially because most parameters produce similar relative embedding errors.

Second, as a result of my analysis I will use the following values in the rest of my thesis: $c_e = 0.1$, $c_c = 0.005$, $t = 0.005$, and $e_t = 30$ min. These recommendations have also been published in [EFF09] as general "best practise" parameters. In the analysis I also spotted rare cases of especially poor performance of the Vivaldi algorithm family. I therefore recommend using conservatively low parameters to also safeguard against such cases.

Third, I could not reproduce the claimed drastic superiority of the Pyxida algorithm proposed by Ledlie et al. [LGS07]. The only effect I could reproduce was a faster convergence, which I consider not as important as stability. In the uniform adaption case Pyxida is more stable to changed parameters, the comparison between a small decay parameter e_t and ∞ had comparable results. In the continuous adaption case, the stability was worse with $e_t = \infty$. The improved convergence speed comes at a cost.

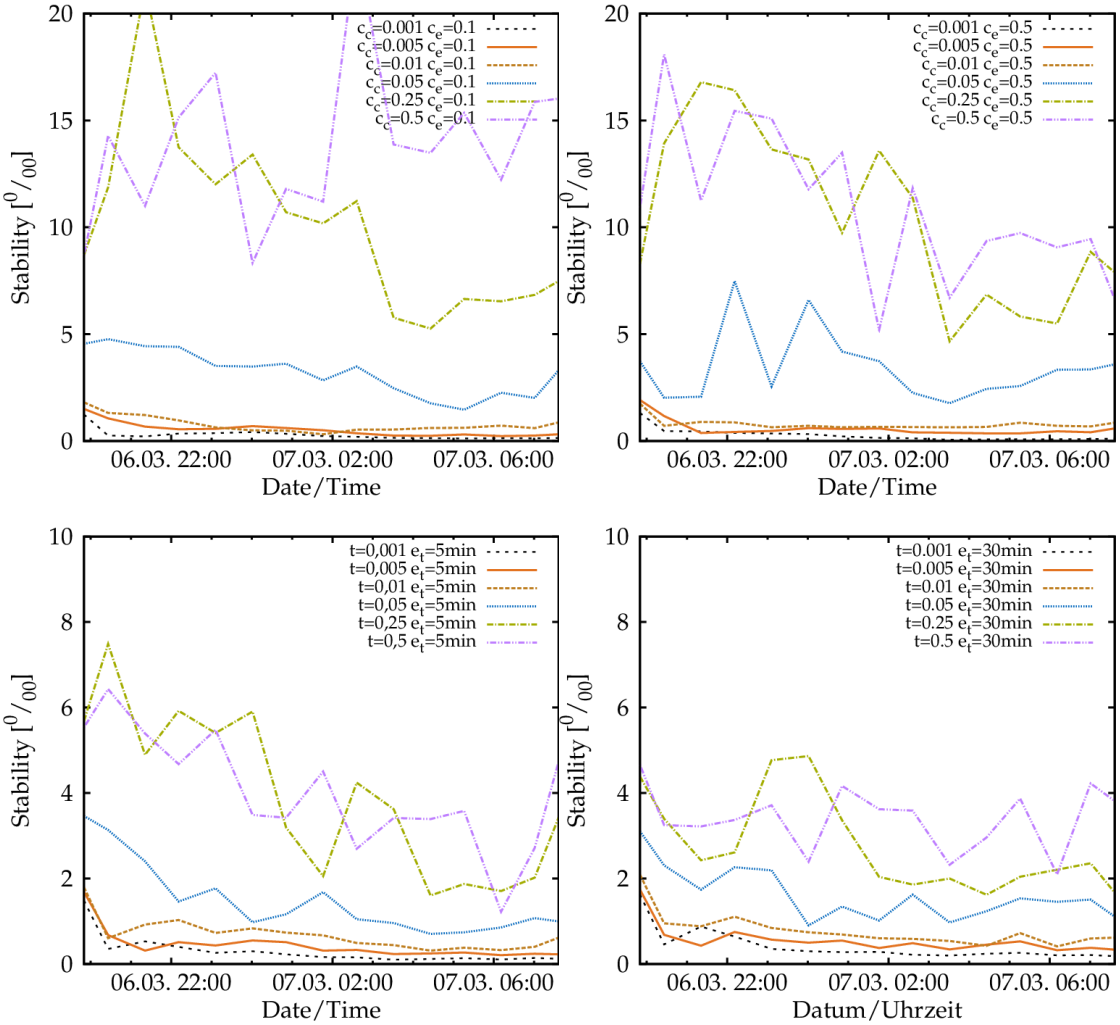

Figure 7.20.: Stability for dynamic trace simulations, using a RTT median filter. Top: Vivaldi algorithm. Bottom: Pyxida algorithm.

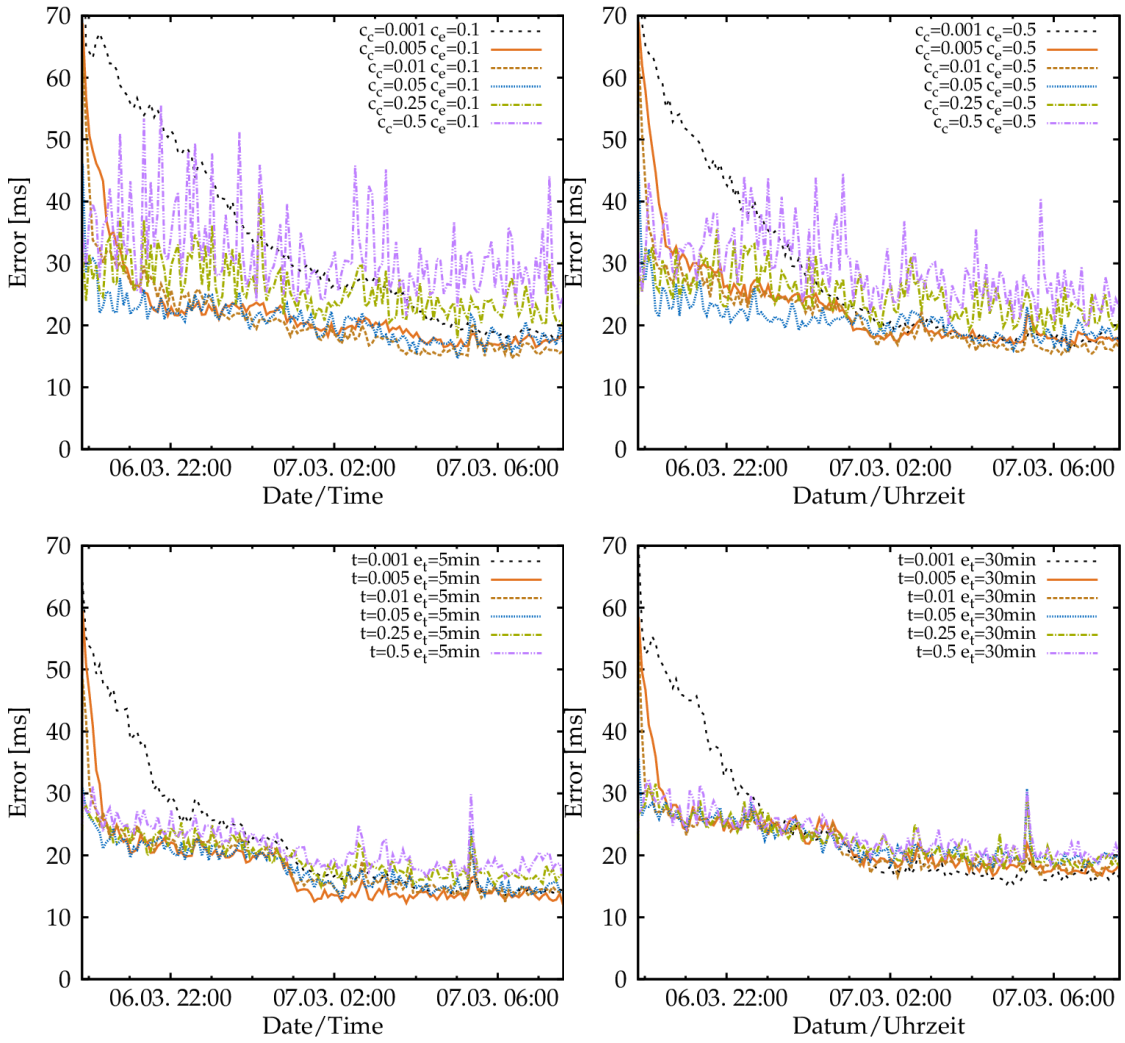

Figure 7.21.: Median error for dynamic trace simulations, *not* using a RTT median filter. Top: Vivaldi algorithm. Bottom: Pyxida algorithm.

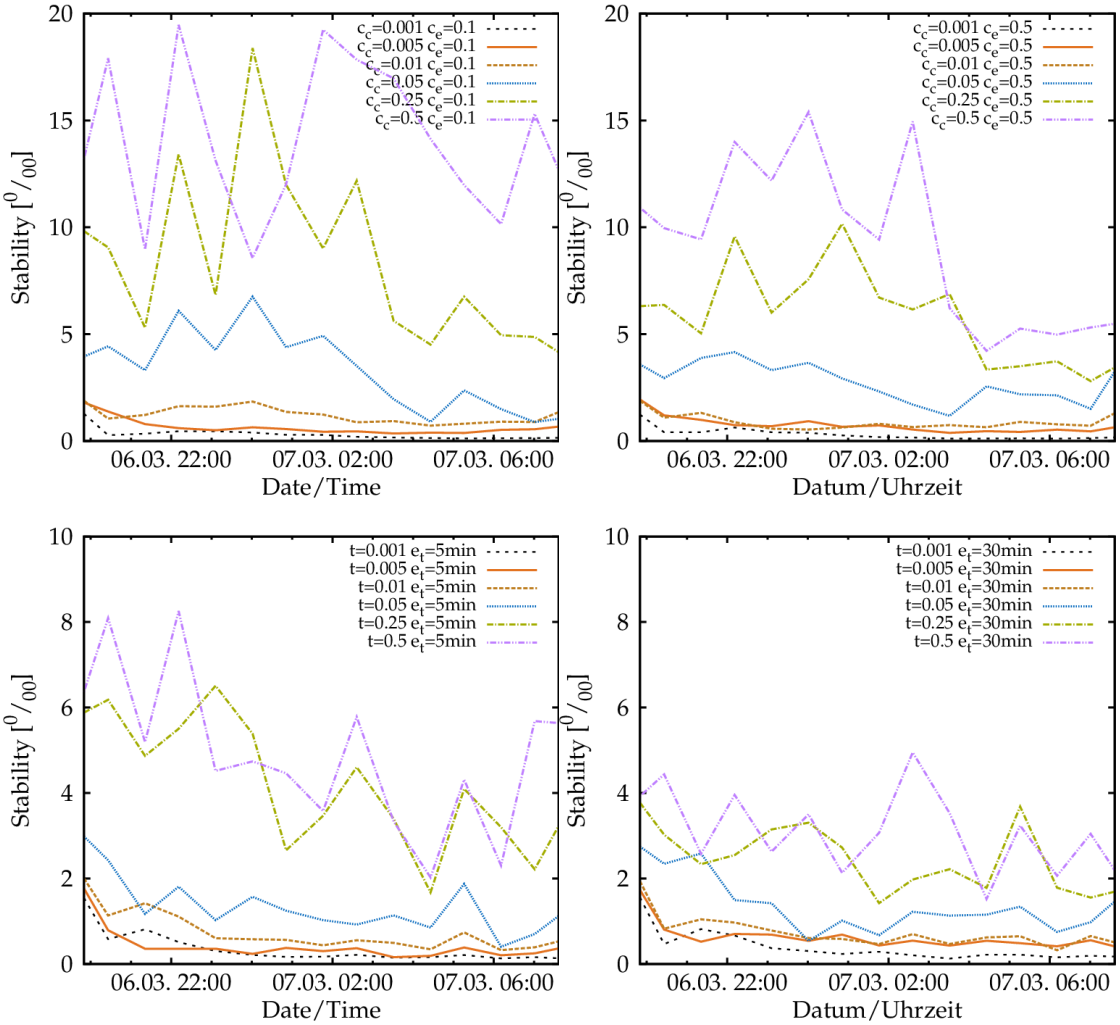

Figure 7.22.: Stability for dynamic trace simulations, *not* using a RTT median filter. Top: Vivaldi algorithm. Bottom: Pyxida algorithm.

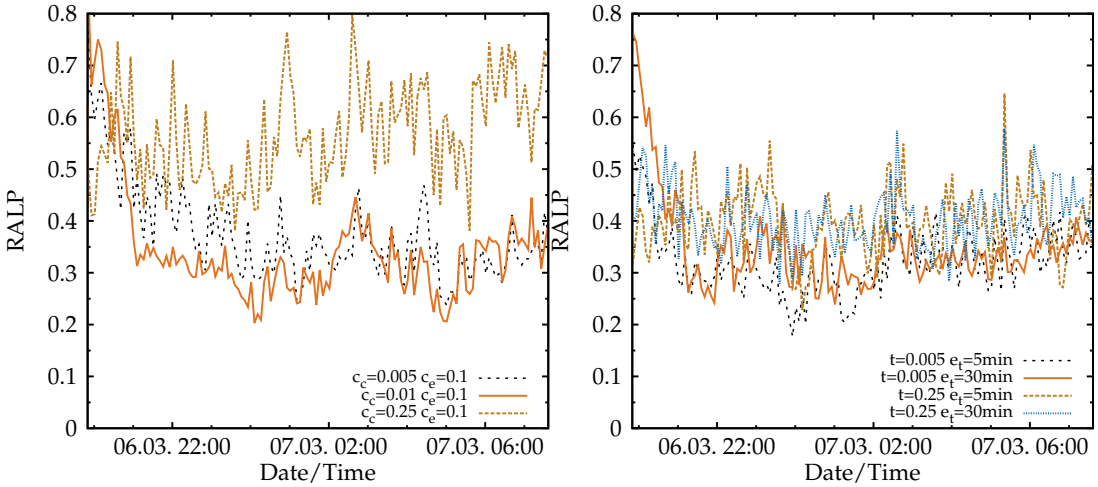

Figure 7.23.: RALP for Vivaldi (left) and Pyxida (right) for the dynamic data trace.

Pyxida reserves buffer space for its round based operation mode. The higher e_t is chosen, the more buffer is needed. However, some applications will not do that trade, like for example the Vuze client, that recently excluded Pyxida. Therefore, choosing the correct parameter for Vivaldi is even more important.

What struck me at most in this study were the large differences between the data sets. Even when comparing static data among each other, there are significantly different reactions of the different algorithm variants. The differences between the results from the static RTT matrices and the dynamic RTT traces were even greater. Ledlie et al. [LGS07] argue that this depends on the magnitude of RTTs that the algorithm receives. I could not confirm this hypothesis.

7.7. Hierarchical Vivaldi Simulation Results

The previous section analyzed the behavior and performance of the Vivaldi algorithm and its Pyxida optimization. Both algorithms were simulated using different parameter combinations. From these simulations I could derive a clear recommendation about the parameter choice to the reader.

In this section, I will introduce the results of my own proposal, the Hierarchical Vivaldi Algorithm. To evaluate the performance of that *hierarchical* algorithm, I ran several simulations and compared their performance to that of Vivaldi and Pyxida. All simulations were carried out using the optimal parameter choices derived in the previous section. I analyzed all the proposed algorithms with the four data sets of both static and dynamic data.

7.7.1. Static data

The algorithm introduces additional parameters that need to be defined before used: The depth of the embedding hierarchy is set to $T_{max} = 2$, or $T_{max} = 1$ when explicitly stated (see figures). Each space $\wp_i$ with $i > 0$ uses a 2-dimensional space plus height component. $\wp_0$ is using 4 dimensions plus a height component. A value greater than 2 for T_{max} was intentionally left out because the resulting communication overhead does not justify the gain. Using $T_{max} = 2$ leads to $\eta = 2^{T_{max}+1} - 1 = 7$ subspaces. A value of $T_{max} = 3$ yields $\eta = 15$ subspaces. With 3 dimensions of coordinates, the achievable gain exceed a reasonable communication overhead. I reassured the behavior of the algorithm with $T_{max} = 3$ in a simulation and it outperformed the $T_{max} = 2$ variant. However that depth needs huge datasets to have a reasonable sized group of well determined nodes and is computational expensive. Therefore, I refrained from carrying out simulations with $T_{max} > 2$. Each simulation was running for 500 seconds.

Hierarchical Vivaldi classifies the nodes' peers into two groups, G_1 and G_2, based on a threshold e_e called the error estimation: G_1 contains the nodes whose error estimation remains under the threshold. To this group, I refer to as *well determined*; G_2 contains the nodes whose error estimation lies above the threshold, which I refer to as *vaguely determined*. I used values of $0.3, 0.5$ and 0.7 for this threshold. Naturally, each stricter threshold shrinks the absolute number of peers that are considered to be well determined, therefore I will refer to the threshold as "picky".

In this evaluation, I studied the following four parameters: Percentage of nodes in the well determined group G_1, RALP for G_1, relative error of embedding in G_1 and G_2. Remember that computing a measure for $G_{\{1,2\}}$ means considering only nodes in the same group in the computation. For example, computing RALP for a node i means selecting nodes that are not already neighbor of i and belong to the same group of well or vaguely determined nodes.

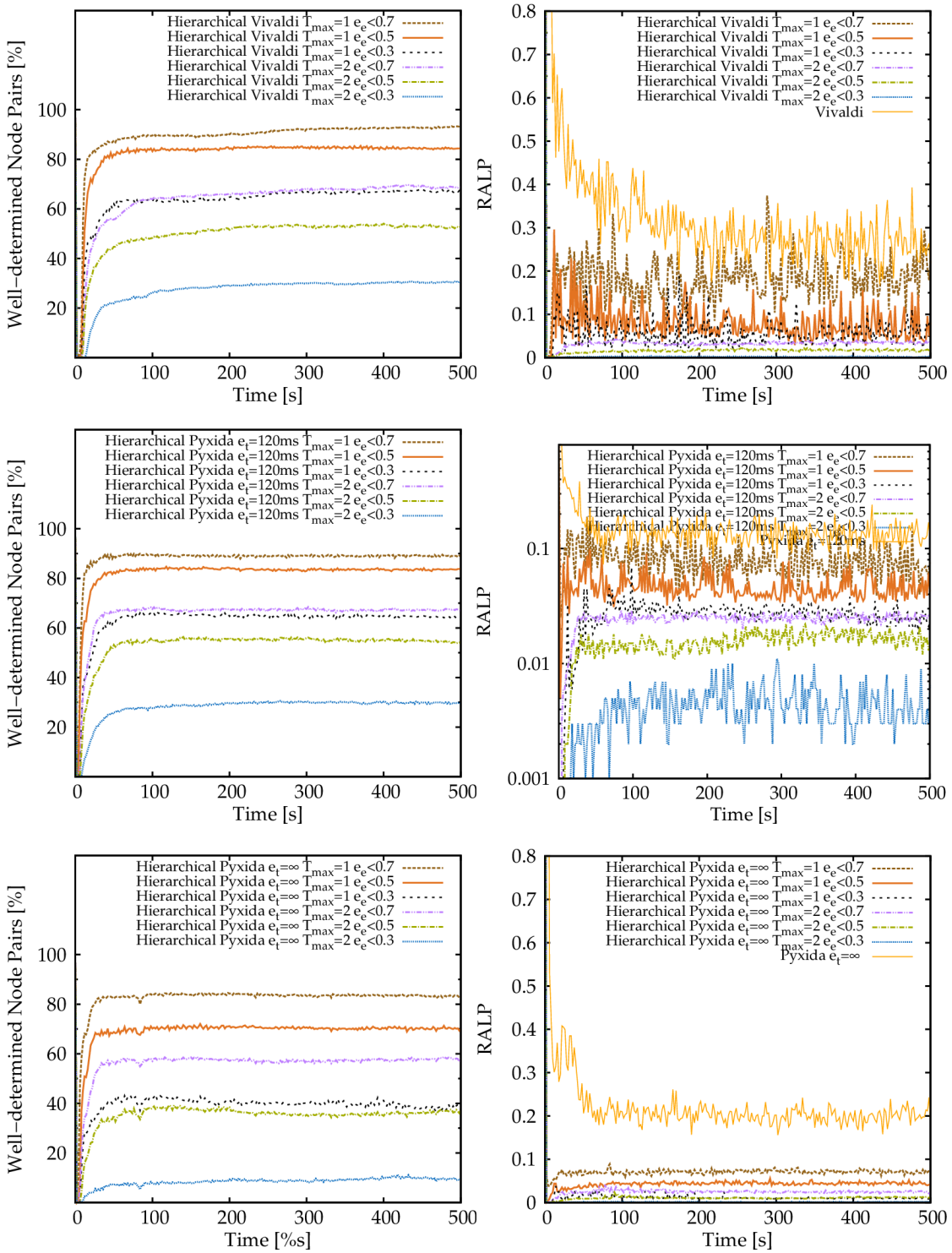

Figure 7.24.: Comparison of Hierarchical Vivalid and Hierarchical Pyxida using the Azureus data set. Left: Percent of nodes in the well determined group G_1. Right: RALP values for well determined nodes.

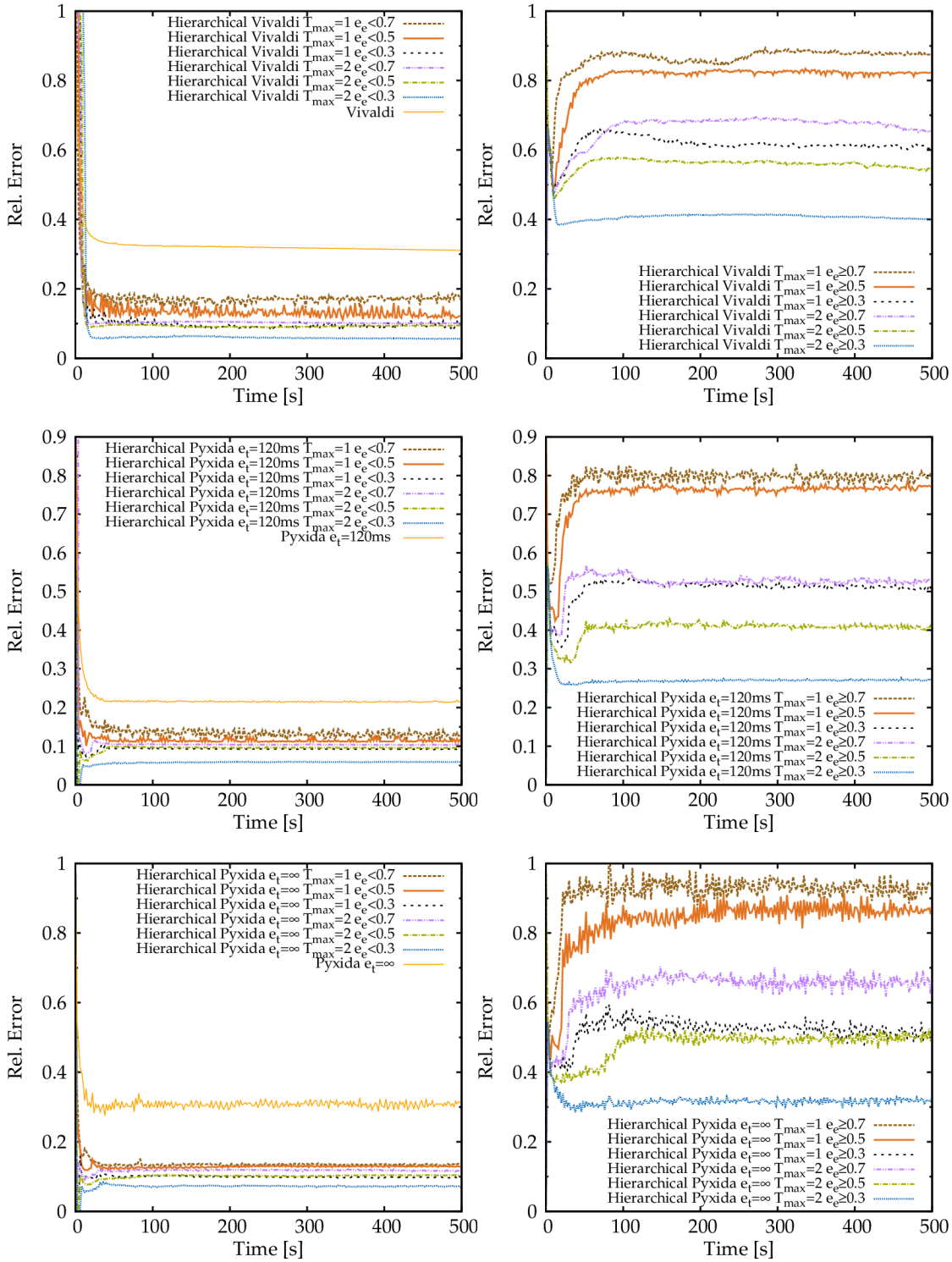

Figure 7.25.: Relative error for well determined (left) and vaguely determined node pairs. Azureus data set.

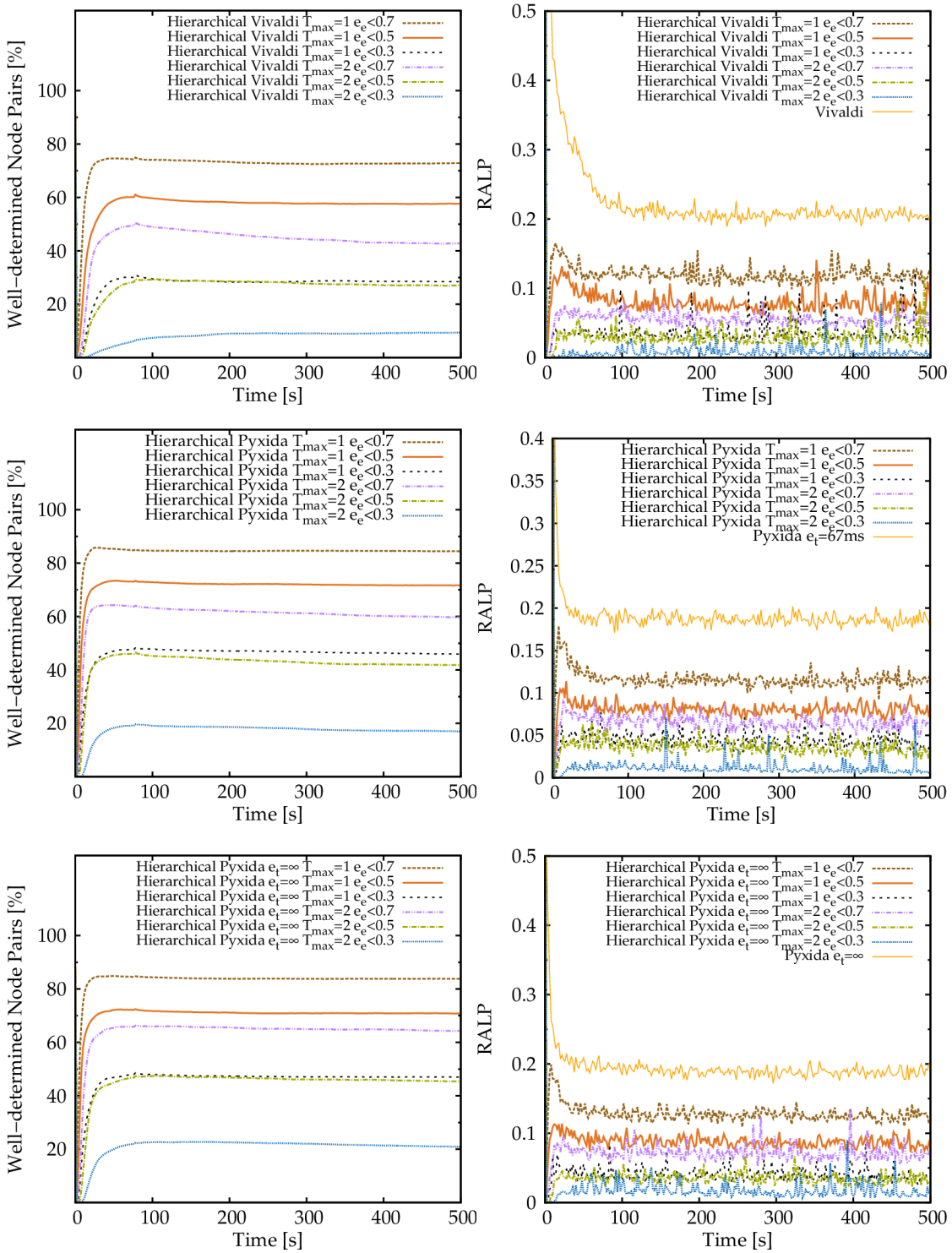

Figure 7.26.: Comparison of Hierarchical Vivalid and Hierarchical Pyxida using the King-Blog set. Left: Percent of nodes in the well determined group G_1. Right: RALP values for well determined nodes.

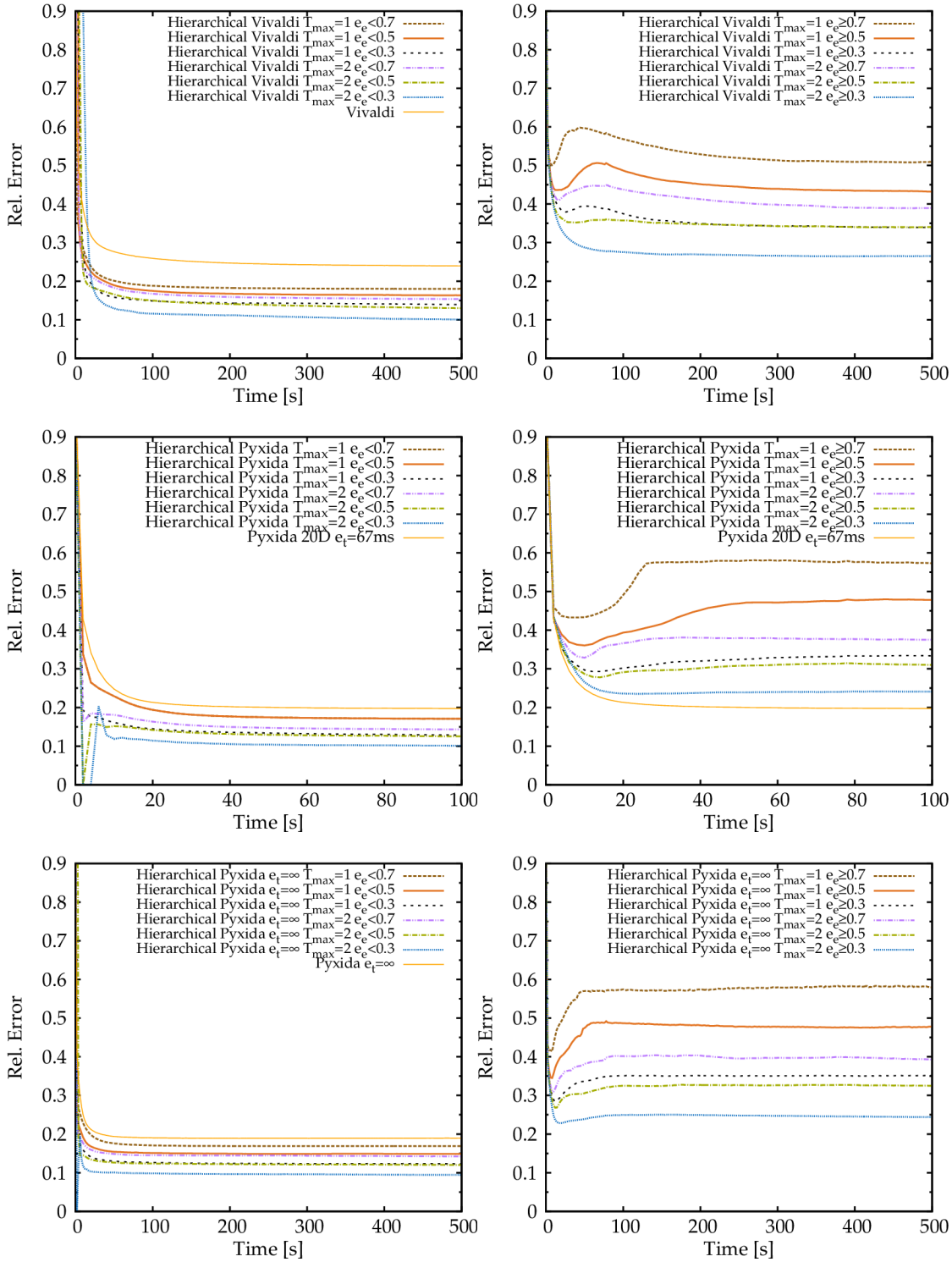

Figure 7.27.: Relative error for well determined (left) and vaguely determined node pairs. King-Blog data set.

Figure 7.24 and Figure 7.25 show data from the Azureus data set, while Figure 7.26 and Figure 7.27 show data from the King-Blog data set. Each simulation was carried out with different parameters e_e and T_{max} for both Hierarchical Vivaldi and Hierarchical Pyxida. The first figure in a data set introduces the percentage of well determined nodes on the left side and the corresponding RALP values on the right. The second figure compares the relative error of the well determined nodes with the relative error of the vaguely determined nodes. Naturally, a more picky threshold classifies more well determined nodes into the vaguely determined group, which leads to better values in that group as well.

What is apparent from the figures is that Hierarchical Vivaldi / Pyxida outperforms its non-hierarchical counterparts clearly. Even when using the most picky threshold each group is filled with candidate nodes, and the pickier the threshold, the more accurate results emerge. The results slightly differ between the different data sets.

The King-Blog data in Figure 7.27 (right) shows the RALP values that Hierarchical Vivaldi (top) respective Hierarchical Pyxida achieves for the different parameter choices. For easier comparison, the figure again shows the RALP of their non-hierarchical counterparts. As one can see, Hierarchical Vivaldi improves the RALP significantly. While Pyxida predicts the latencies only within a margin of error of about 20%, the hierarchical variant can achieve a margin of error as low as a few percent only. In general, a second-order error embedding, i. e. $T_{max} = 2$, outperforms a first-order embedding.

In contrast, the Azureus data using Hierarchical Pyxida (Figure 7.24, bottom), only ten percent of nodes are in the well determined group, which translates roughly to 25 nodes in the examined Azureus data. The corresponding RALP values became too unstable to make any reasonable assumptions, therefore it is not included in the plot. This is, however, not a defect of Hierarchical Vivaldi because that data set already performed poorly using the $e_e = \infty$ setting in the previous non-hierarchical setup, cf. Figure 7.23. Naturally, the values grow more unstable with a much smaller group of nodes in the hierarchical case. For the ten times larger King-Blog data in Figure 7.27, the same setting produces good results, as discussed previously.

Moving on to the relative error results, the left column of the each figure shows the error of the well determined nodes. Again, the hierarchical variants clearly outperform their counterparts in both data sets. In the right column, the error of the vaguely determined nodes leads to the best performance, when the threshold is set to the picky $e_e < 0.3$ value, because nodes with good relative errors are rejected from the best group and decrease the relative error of the vague group.

From all figures, the impact of the depth parameter T_{max} is clearly obvious. Best results are achieved using a depth of 2. With $T_{max} = 1$ and $e_e = 0.3$ the relative error sometimes meets or outperforms a $T_{max} = 2$ setting with a undemanding e_e value of 0.7. That parameter choice is therefore an appropriate choice, where communication overhead is critical. Hierarchical Vivaldi reaffirms another observation from the non-hierarchical simulations. The Pyxida variant matches but does not outperform the Vivaldi variant

when using hierarchies, hence Pyxida's resource demands are not justified, because Vivaldi can be used instead.

The absolute improvement strongly depends on the setting where the algorithm is applied. When there are sufficiently many peers in the system, we can set a very picky threshold and thereby reduce the RALP below one percent. A generous threshold ($e_e = 0.7$) with a first-order error embedding ($T_{max} = 1$) keeps more than 80% of the nodes in G_1. A picky threshold ($e_e = 0.3$) with a second-order error embedding ($T_{max} = 2$) only keeps about 20% of the nodes in G_1. Since King-Blog contains 2500 nodes, this means that G_1 contains between 500 (picky) and 2000 (generous) nodes. Generally, I found that the error estimator e_e is much more accurate than Vivaldi's built-in local error. As one can see, the proposed Hierarchical Vivaldi algorithms outperform the plain Vivaldi algorithms by an order of magnitude under ideal circumstances.

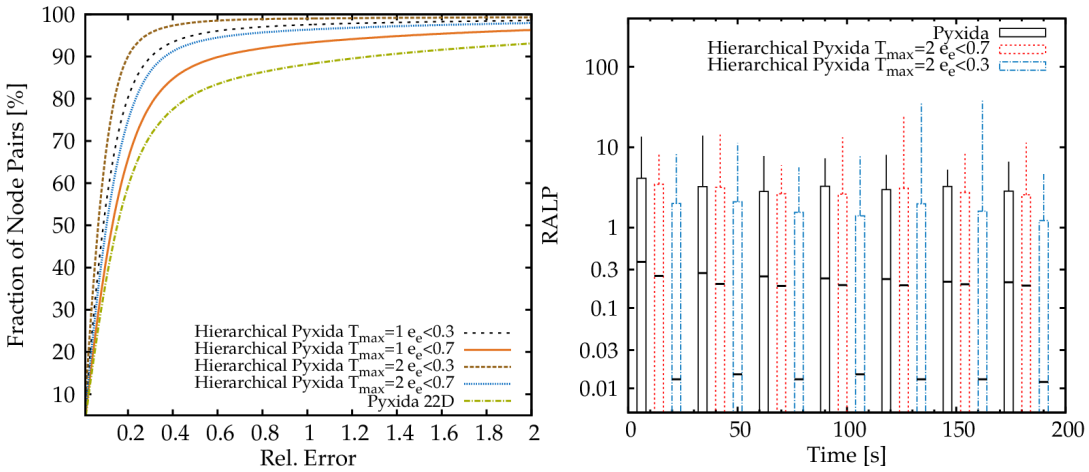

Figure 7.28.: Relative error (left), RALP (right) snapshots at simulation end.

Figure 7.28 sums up the static simulations. The figure to the left illustrates the effect that this selection has on the latency estimation error. For all parameter settings, Hierarchical Pyxida reduces the estimation error for the nodes in G_1 simply by selecting those nodes that have a low estimation error. The error for nodes in G_2 is significantly worse.

The box plot in Figure 7.28 provides an in-depth RALP analysis of the King-Blog simulation at various points in time during 200 seconds. Clearly, the well determined group, which was picked by a 0.3 threshold using $T_{max} = 2$ proves to lead to the best range of RALP values across the data set.

In general there are only slight differences between the Hierarchical Vivaldi and Pyxida variants.

7.7.2. Dynamic data

In the previous section I compared non-hierarchical simulations using static and dynamic data. This section completes the Hierarchical Vivaldi analysis, by comparing my

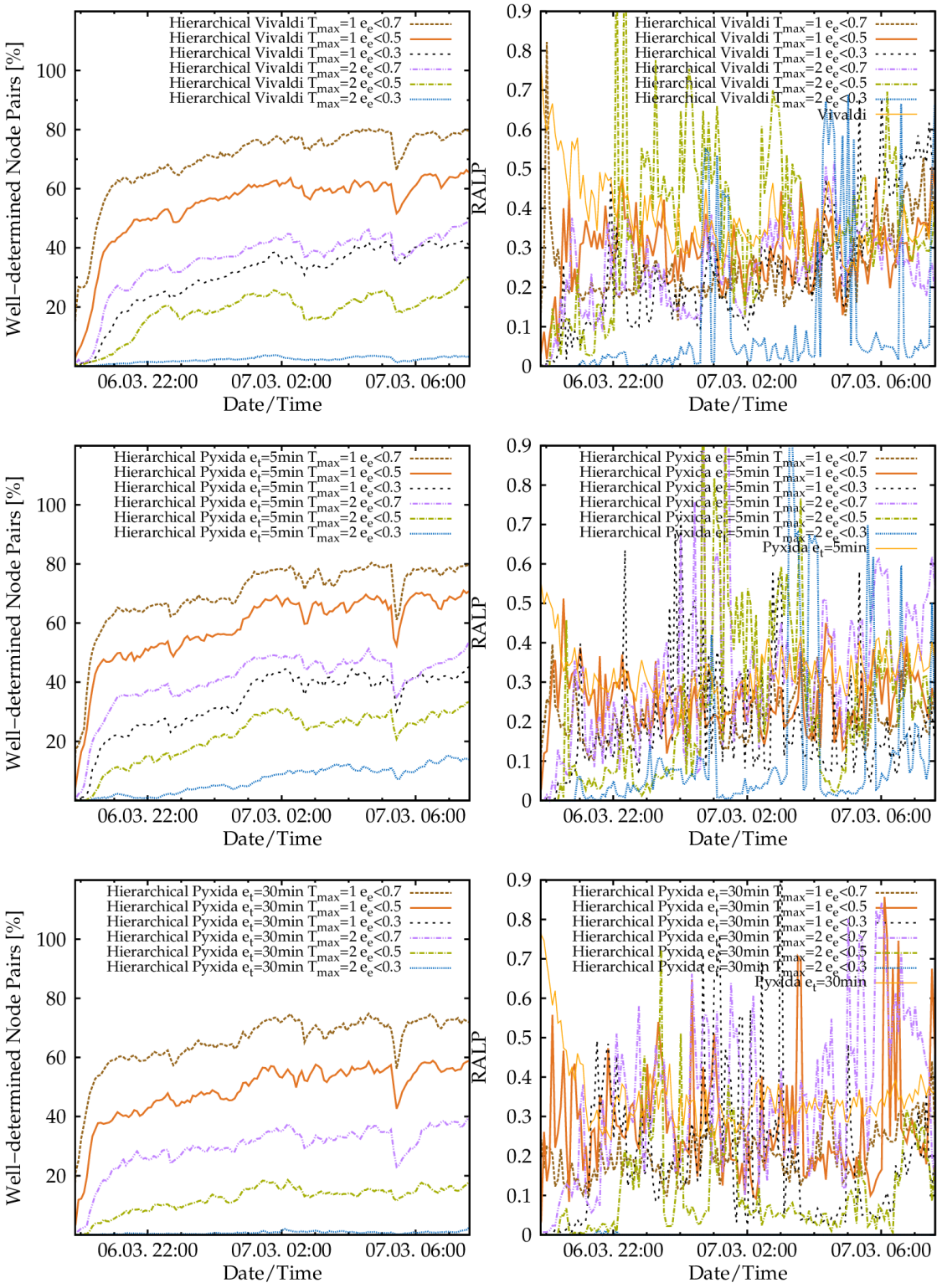

Figure 7.29.: Comparison of Hierarchical Vivalid and Hierarchical Pyxida using the dynamic data set. Left: Percent of nodes in the well determined group G_1. Right: RALP values for well determined nodes.

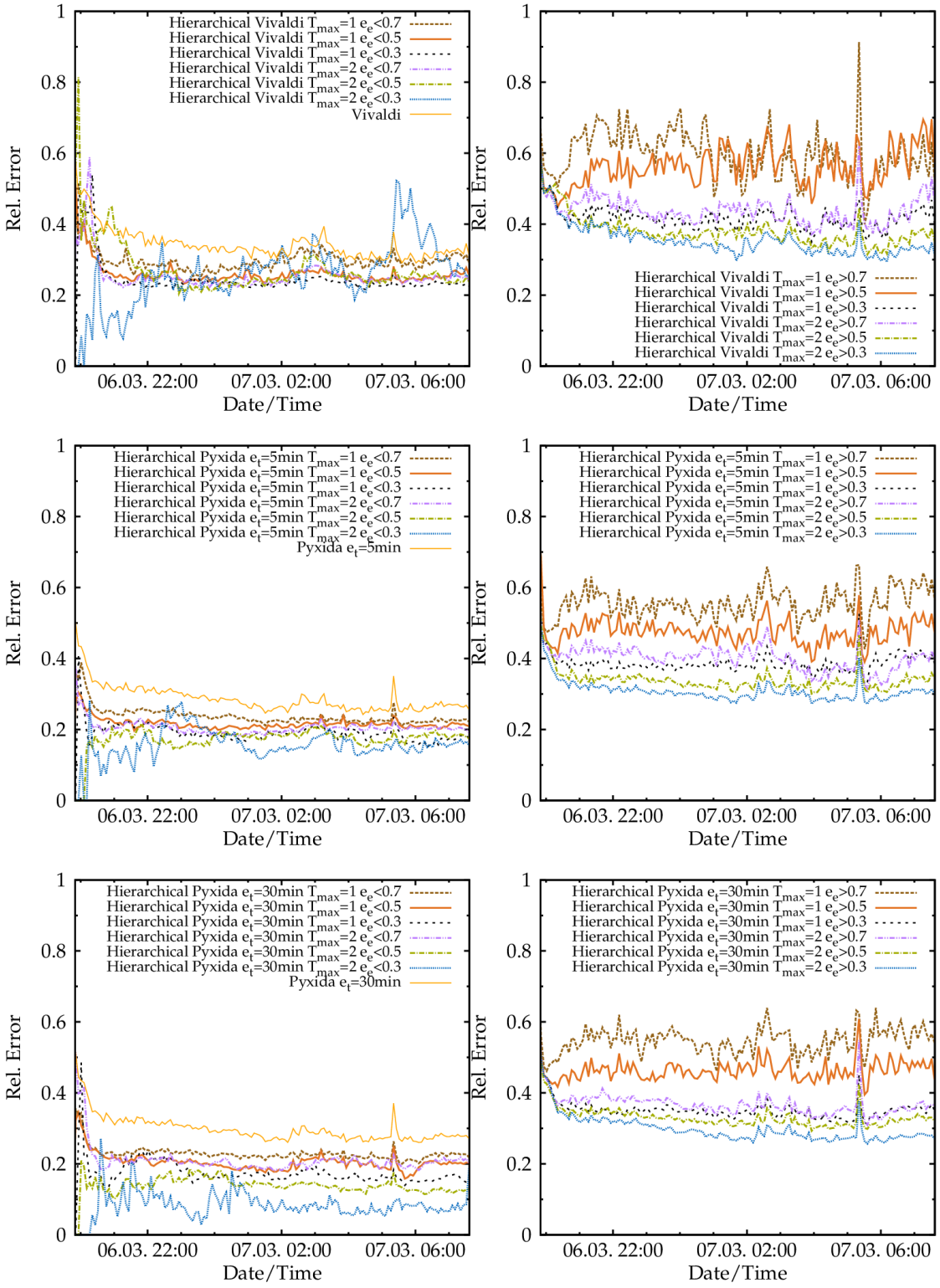

Figure 7.30.: Relative error for well determined (left) and vaguely determined node pairs. Dynamic data set.

previous static Hierarchical Vivaldi analysis using dynamic data

Figure 7.29 introduces the percentage of well determined nodes in the left column and the RALP values on the right. One can easily see that the data set is not suitable for a RALP computations: When using $T_{max} = 2$ and the most picky e_e value, Hierarchical Vivaldi has not even 2 % of nodes in the well determined group. For a total of 89 nodes a computation of RALP is impossible. Although the values indicate satisfying results, they are just too noisy. However, the computation of the relative error does not involve a neighborhood of peers, therefore Figure 7.30 is more significant. All members of the well determined group exhibit smaller error values, than the original non-hierarchical counterparts. The already mentioned most picky setting exhibits some disturbances that I account to the small number of peers.

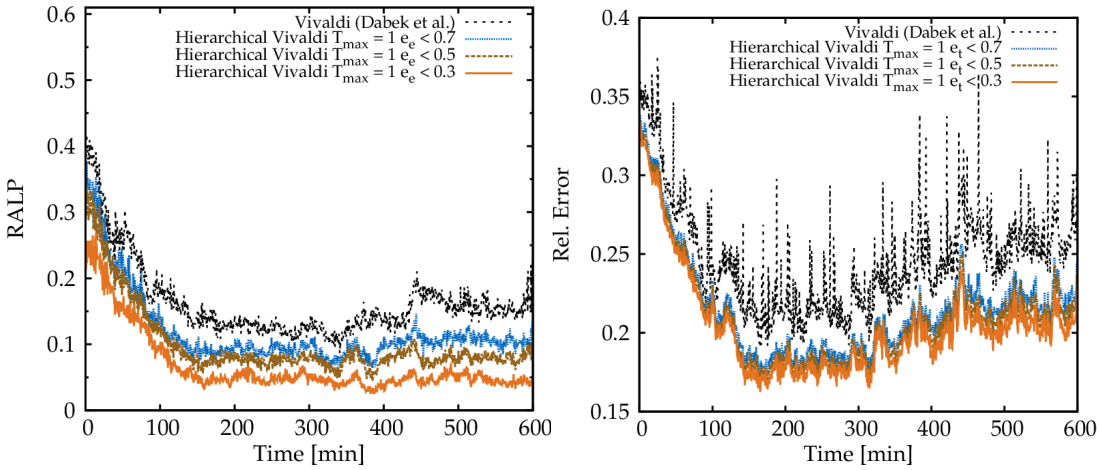

Figure 7.31.: Relative application level penalty (left) and relative embedding error (right).

7.7.3. Conclusion

After the promising results from the static simulations and the noisy but satisfactory results in the dynamic trace, I evaluated my proposal on PlanetLab using a maximum number of nodes online. I restricted the measurement to a short time frame to ensure a low probability of loosing a node due to hardware or software failures. Failures on PlanetLab happen quite often, which is the reason for the dynamic trace's inferior size. I built Hierarchical Vivaldi into the IGOR overlay daemon and bootstrapped it with a maximum of 32 neighbors per node. To save bandwidth only $\eta = 3$ spaces were used. Figure 7.31 shows the relative application level penalty for an overlay with 312 peers for about 10 hours. (Figure 7.31 shows the respective relative embedding error, and Figure 7.32 shows the fraction of well determined node pairs.) Upon start-up, the overlay topology is quite sub-optimal, which leads to large RALP values. Over time,

the peers settle in the network coordinate system, and the RALP values improve. After about 1 - 2 hours the overlay has settled completely.

These results reaffirm the results from the simulations I presented in this chapter. They show that Hierarchical Vivaldi outperforms the classical Vivaldi with about a factor of 3, still a significant improvement and an expected result with $T_{max} = 1$. The overhead of running a real KBR overlay with structural constraints and high churn rate of neighbors shows no negative impact. Furthermore, all simulations proved one important point: The pickier a threshold is, the smaller and the better selected is G_1. Hierarchical Vivaldi is able to assess the quality of an embedding. Therefore, these results show that Hierarchical Vivaldi works and is a real improvement over classical Vivaldi. Furthermore, the results demanded further research with a maximum number of peers and a real world experiments. In the tradition of network coordinate research I contacted the Vuze developers and thus gained such a real world setting. The results are presented in the next chapter.

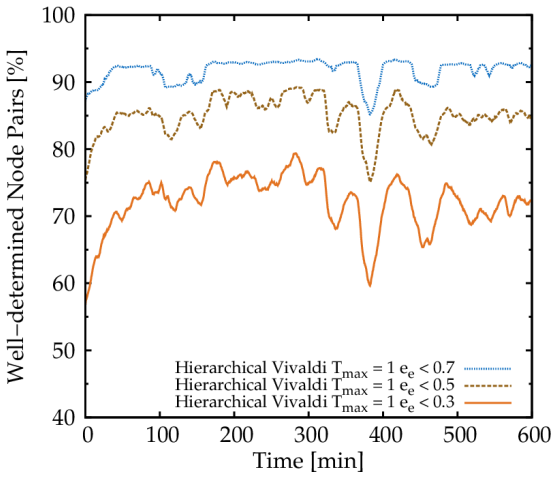

Figure 7.32.: Fraction of well determined node pairs (measurement).

8. Hierarchical Vivaldi Deployment in a Planetary-Scale Overlay

The previous chapter gave the reader a glimpse of the performance of Hierarchical Vivaldi. The results of the simulations were promising, some of them outstanding. However, a simulation using a captured dynamic trace is too short on input data to get a more realistic view of the algorithm. Above all, real world measurement data is known to be superior to testbed data [CB10]. A first peek at an online measurement in the conclusion of the previous chapter affirmed my findings on Hierarchical Vivaldi's performance. A planetary scale experiment introduced in this chapter will reassure the findings once more: A Hierarchical Vivaldi plug-in was included into the Vuze client (cf. Section 3.4). That client is extremely popular among users download.cnet.com counted approximately 8.2 million downloads [Int11] in October 2011. Using a voluntary statistic collection component built-into the plug-in together with its enormous user base, I was able to gain unique insights into both Vivaldi and my proposed algorithm.

This chapter briefly introduces the infrastructure that was used for collecting data, before presenting an in-depth evaluation of Hierarchical Vivaldi. Previously, network coordinates were said to only be able to distinguish among continents but not among autonomous systems, countries, or even cities. The unique viewpoint along with experimental data that I have been able to gather in one of the world's largest BitTorrent networks allows me to retrieve the network coordinates´ honor. It shows that Vivaldi-like algorithms can very well predict round trip times and can distinguish autonomous systems or geographic regions. The analysis marks the final building block of my thesis.

8.1. Methodology

Before presenting results of the evaluation, this section briefly outlines the set up of the study: It discusses the infrastructure, the study's design, the plug-in itself, and the different measurement settings receive attention.

8.1.1. A Large Scale Overlay

The goal of this real world measurement was to assess the performance of an improved Vivaldi algorithm in a realistic setting. Steiner et al. [SB09] pursued a similar goal. They evaluated the performance of the built-in Vivaldi algorithm in Vuze. Therefore, they built a crawler that collected data from the Vuze BitTorrent overlay. The actual crawling was

done from different nodes inside the network including the Eurecom institute, France and Mannheim, Germany. Unlike Steiner, this measurement component was included as a plug-in into the official distribution of Vuze. Thereby, one can study the overlay from the clients' perspective rather than having an outside view only. Focusing on different clients, different vantage points. One could for example, study the well-connected peers in university networks and compare them to peers in DSL networks.

The measurement setup differentiates between so-called *reporter nodes* and *plain nodes*. Reporters actively participated in the study and provide a vantage point. They regularly reported to the collector site located at the Technische Universität München. All other Vuze nodes ran the Hierarchical Vivaldi algorithm as well, but they did not report to the collector. Each node executed the stock Vuze Vivaldi algorithm using two dimensions plus height component (Vivaldi 2D+H) in parallel to the Hierarchical Vivaldi algorithm.

An ordinary Vuze node only became a reporter when its user explicitly enabled the reporting mechanism inside the client. The reported data contained the coordinates of the reporter and its current peers as well as the RTT to these peers (measured at the application layer, cf. Figure 3.5). Furthermore, the meta-data included the autonomous system ID and geo-location of the peers (as derived from the peers' IP addresses), and the peers' unique IDs. It did not contain any peers' IP addresses or any torrent information. For privacy reasons, we also did not store the IP addresses of the reporter nodes.

Once per hour, each active reporter connected to the collector site and sent reports about all its peers. To promote the interconnectedness of reporter peers, the reporter received a list of other recent reporters. That list was composed using 25% of the most recent reporters (during the last hour) in an uniform random way. In the time from May 2010 to March 2011, the collector site thereby received $3.883 \cdot 10^{10}$ reports from 5 013 unique reporters. An average report contained up to one kilobyte of data and described one Vuze node as seen from the reporter. A load balancer distributed the reports across six collector machines so that the system could handle peak times without data loss.

8.1.2. Vuze Plug-In

In Section 3.4, I introduced the Vuze BitTorrent client. Vuze is popular among users and has an impressive history among research. The client is one of the first ones that included the original Vivaldi algorithm. Scalability issues with Vivaldi became apparent and were reported from developers of that very client. Being contacted from Vuze developers, Ledlie et al. at Harvard conducted the first large scale analysis of Vivaldi using Vuze. Their research resulted in the proposal of Pyxida [LGS07]. Congruently, Choffnes et al. chose to implement Ono [CB08] as a Vuze plug-in. The steady use of Vuze as a scientific device especially for proximity research is also reflected in its API. The `GENERIC_NETPOS` interface allows seamless integration of position providers (**class** `DTNetworkPositionProvider`) into the client. Vuze is able to handle different position providers simultaneously. Just like the ID interface in IGOR (cf. Section 4.4) each position provider's payload that is attached to a message is serialized and prepended with an

identifier. With the help of this ID, each client can call the corresponding module, if it is registered.

The implementation details of the plug-in are available in [Fra11]. Here I introduce only the modus operandi. On startup and shutdown the Hierarchical Vivaldi plug-in restores and saves the node's most recent coordinates. Reusing saved coordinates allows me to investigate the reusability of coordinates (cf. Section 8.2.1).

As previously mentioned, the plug-in included opt-in functionality, which enabled the reporting of statistics to the data collector. For privacy reasons I avoided storing IP addresses of users. Measurement data was passed to the system via the HTTP protocol. Therefore, the data collector was the single point of the system, where the IP addresses of reporters accumulated. These addresses were never written to disk, but were kept in a recently used (lru) list in memory, to send an excerpt of that list to a reporter upon contact, in order to create locality upon the reporting nodes. Hence, measurements between those nodes could be studied from both nodes. That node could then ping the other client via a call to `DistributedDataseContact::import(InetSocketAddress)` and `DistributedDataseContact::ping()`.

To dynamically reconfigure the plug-in during the experiment, the design included a command and control component. The remote override could alter parameters of the Hierarchical Vivaldi algorithm and parameters like the number of pings to other overlay nodes. That part could be disabled by the user, separate from her initial admission to data reporting.

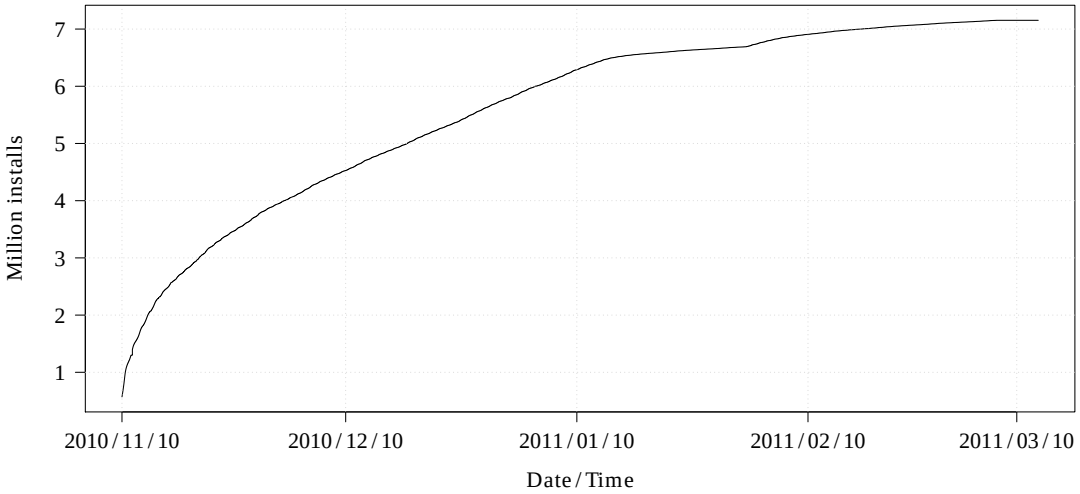

Figure 8.1.: Total downloads of the final plug-in version.

8.1.3. Chronological Sequence

The experiment was divided into 4 phases. Initially, smaller tests on PlanetLab were carried out. With the first inclusion into the Vuze code base, only beta testers for the 0.45 version of the client received that functionality. They joined the 500 PlanetLab nodes that already served as testbed for the plug-in development and reporting infrastructure. This marks the first phase of the experiment.

The experiment started officially with the first release of the Hierarchical Vivaldi plug-in May 2010. The official release of Vuze 0.45 introduced the Hierarchical Vivaldi algorithm to a wider audience by marking it as a recommended plug-in. Recommended plug-ins gets installed automatically by the Vuze extension mechanism. The release marks the second phase of the experiment. The broad user base sparked participation among users. Up to 1000 activations of the statistics component were measured shortly after the release. Therefore, the PlanetLab nodes were suspended, until September 2010.

At this point, we re-enabled the PlanetLab reserve in a third phase, to investigate their impact on the system performance.

In a final phase in November we disconnected the latter once again, to measure a pure end user based network (cf. Section 8.2.1). This was done to ensure unbiased results, as PlanetLab features superb network connections, not found in end user homes. The final version 485 of the plug-in was downloaded 7 million times, as Figure 8.1 shows. The official release of Vuze 0.46 in January 2011 marks the end of our experiment and Vuze removed the plug-in. Figure 8.2 sums up the lifetime of the Hierarchical Vivaldi plug-in based on the number of reporter peers. The color code at the bottom of the figure indicates the presence of PlanetLab nodes.

Pyxida filters and smooths the RTT measurements before they are entered into the model. However, as I mentioned previously that algorithm was removed from the Vuze client. Therefore, I was not able to test the Hierarchical Pyxida variant in practice, using the Vuze client. Nevertheless the simulations from the previous chapter did not show huge differences between Hierarchical Pyxida and Hierarchical Vivaldi.

8.1.4. Hierarchical Vivaldi Parameters

The same settings, as in the previous simulation chapter apply for the experiment. Hierarchical Vivaldi and the original Vivaldi algorithm use the previously recommended parameters $c_c = 0.1, c_e = 0.01$, and the depth parameter $T_{max} = 2$.

For Hierarchical Vivaldi, the topmost space $\wp_0$ used 5 dimensions, while all other spaces $\wp_{1...6}$ used 3 dimensions. Vivaldi was measured using both a two dimensional space plus height component and a 5 dimensional space. Due to data loss on one of the collectors´ nodes, the experiments lacks 1 month of measurement data for the 5 dimensional Vivaldi.

Compared to the simulations in the previous chapter I adapted the used measures slightly: Both the RALP and the relative error measure (equations 7.4 and 7.2 on page 80)

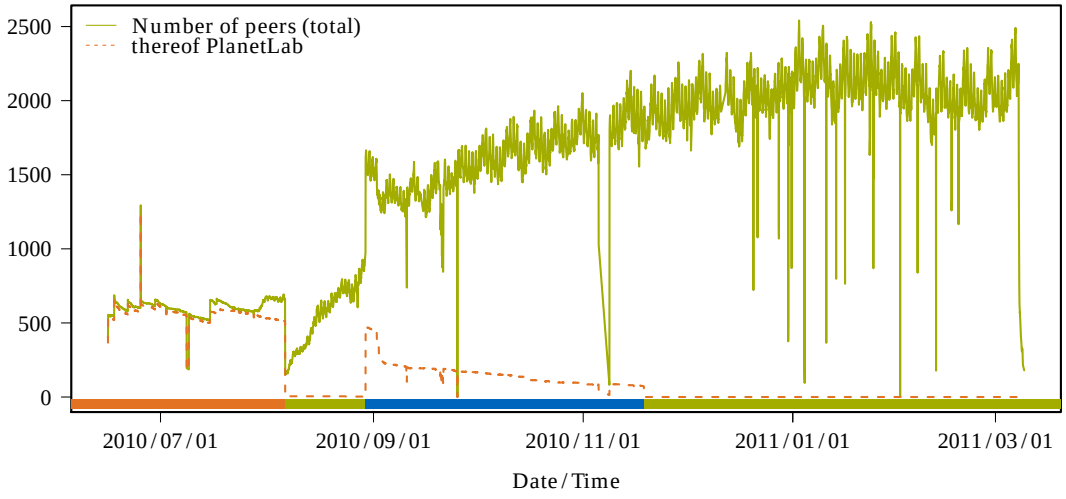

Figure 8.2.: Absolute number of Peers contributing statistics at least once per hour. Bottom-Colors indicate:
- Only PlanetLab nodes
- Only end user nodes
- PlanetLab and end users

are changed from a $\frac{1}{n^2}$ to a $\frac{1}{n}$ factor. This quadratic coefficient adapts the measure to the number of measurements in a full mesh. However, the Vuze measurement setup is only sparsely connected. Therefore, I relaxed the quadratic adaption.

Another restriction is the scope of the RALP measure. Previously, I computed RALP using *non*-neighboring nodes. This means that, both nodes have not adapted their coordinates to their mutually measured RTT recently. However, this experiment abandons that modus operandi because measurements are carried out using the Vuze framework, as previously described. Using the built-in `ping()` adds the target node to the pool of potential neighbors. A ping reply mandatory involves adapting the own coordinate. Including a separate measurement component would have been an intrusive move, which I decided not to take.

8.2. Results

This section presents a summary of findings of my evaluation study. The full analysis is available in [Fra11]. All measurement data is published at the author's website [Els11]. In the first section I compare Vivaldi and Hierarchical Vivaldi. In the second section I use the unique viewpoint from inside the network to analyze the capabilities of a large deployment of a network coordinate algorithm.

8.2.1. Evaluation of Hierarchical Vivaldi

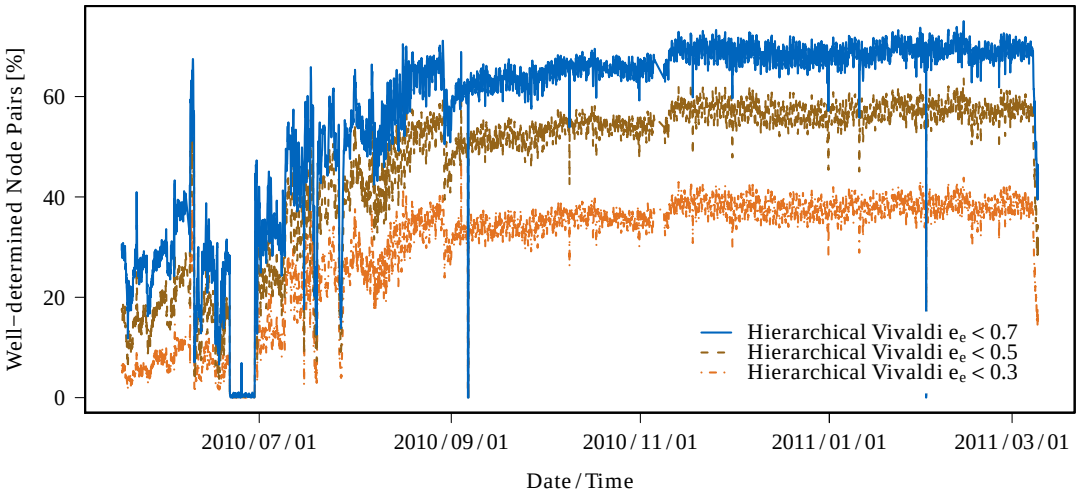

Figure 8.3.: Percent of nodes in the well determined group G_1.

Figure 8.3 complements Figure 8.2 by showing the percentage of nodes in the well determined group over the course of the whole simulation. The figure affirms the findings from my previous simulations. At $T_{max} = 2$ Hierarchical Vivaldi includes 60% of nodes when using a relaxed threshold into the well determined group. A picky threshold includes only 30% of nodes into the well determined group, the static simulations in contrast included slightly less nodes. During the experiment all groups were appropriately filled with nodes, no "under-runs" like in the dynamic simulation happened.

The overall performance of the system was optimal, Hierarchical Vivaldi met its expectations. In Figure 8.4 I present the RALP results of the entire experiment. Disregarding the spikes, caused by system shut downs when e. g. newer versions of the plug-in were rolled out, the performance of the hierarchical variants is superior to the plain Vivaldi variants at any time. From that overview one can quickly get a glimpse of the stability of Hierarchical Vivaldi during the entire experiment. Furthermore, it initially convergences much faster than Vivaldi 2D+H to a low RALP value. Figure 8.5 (left) shows the same data in a CDF view. Clearly, the Vivaldi 2D+H variant is far from accurate. While 90% of nodes have a RALP value of 0.13 using the most picky threshold for Hierarchical Vivaldi, the 2D+H variant includes RALP values up to 0.87. 90 % of the nodes in Vivaldi 5D have a RALP value of 0.22, which is roughly 1.8 times higher than the Hierarchical Vivaldi. Vivaldi 2D+H's performance is unable to reach the performance of 5 dimensional Vivaldi or Hierarchical Vivaldi at any time in the experiment. The surprisingly good performance of Vivaldi 2D+H during August is quickly explained: It marks the

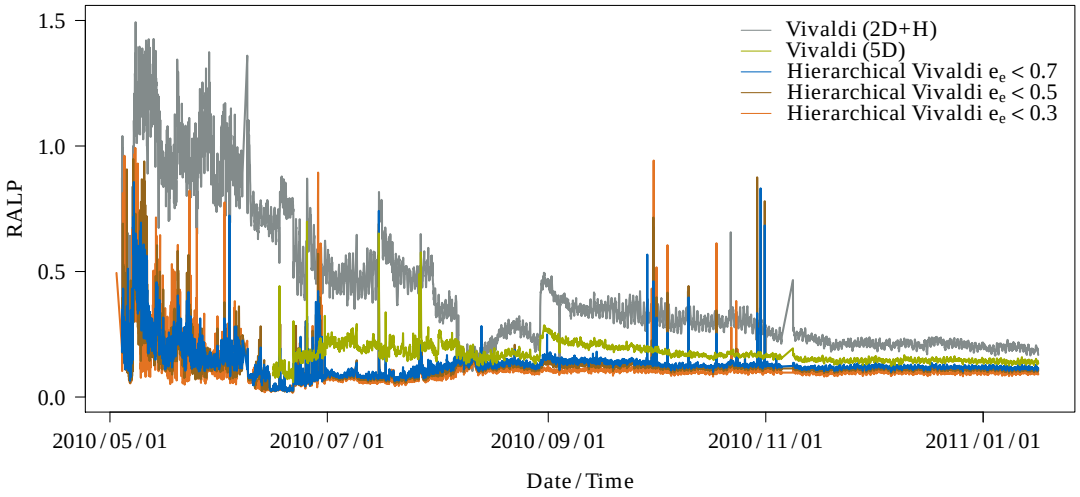

Figure 8.4.: RALP values during the entire experiment.

begin of the second experiment phase. At that time, a huge number of nodes with stable Vivaldi 2D+H coordinates began to report statistics.

However, that data includes large periods of instabilities, e. g. plug-in restarts, node joins, leaves. Therefore, the right part of Figure 8.5 shows the final phase of the experiment from 28.12.2010 until 13.01.2011: After the final release of our plug-in and the final departure of all PlanetLab nodes, the situation does not change radically. That steadiness reaffirms my results: Under ideal circumstances the 2D+H variant reaches a RALP value of 0.2, the 5D variant 0.14 in the 90 percent percentile. Both results are slightly better than the simulation results, however, they are computed using the actual neighborhood, as previously explained. Hierarchical Vivaldi with the most picky threshold reaches a value of 0.1, slightly worse than the simulation results that reached that value using the same T_{max} value, but a threshold of $e_e < 0.7$. The drastic improvement of Vivaldi 2D+H shows that under ideal circumstances, these parameters are able to yield mediocre results. However it is unable to handle joins or leaves of a large number of nodes, as I present in the next section. Hierarchical Vivaldi also shows an improvement. Over the total time of the experiment (Figure 8.5 left), the last ten percent of the hierarchical algorithms seem unable to match Vivaldi's performance. However that data includes the whole range of the experiment. As I previously mentioned, the results for the first month of Vivaldi 5D are lost. Presumably the missing data would increase Vivaldi 5D's RALP value in Hierarchical Vivaldi's favor, because the 5D data misses the unstable initial phase of the experiment.

The relative error for that same period from 28.12.2010 until 13.01.2011 in Figure 8.6 (top) reaffirms my findings once more: The hierarchical embeddings is superior to both non-hierarchical settings. In a similar way, as the King-Blog simulation, from the previous chapter, it reaches a value of 0.1 using the most picky settings. In spite of the

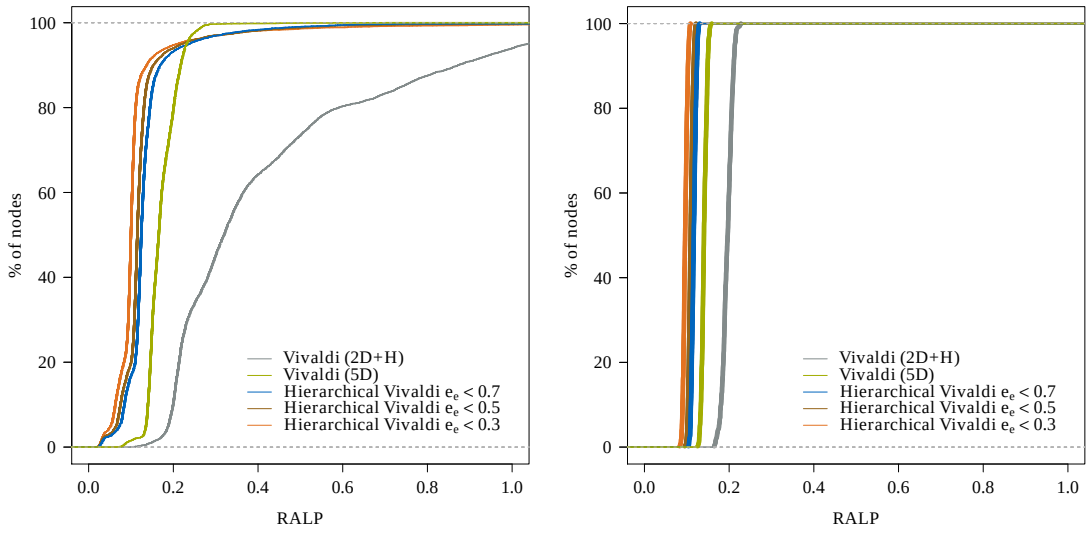

Figure 8.5.: ECDF RALP values during the entire experiment (left) compared to the final experiment month (right).

good performance of 5D Vivaldi, the figure clearly shows the superiority of Hierarchical Vivaldi, in both quantiles and median values. The analysis of the group size in the bottom part of the figure yields no new insights. All group sizes confirm the findings from the previous simulations.

The real potential of Hierarchical Vivaldi has so far not been accessed. Figure 8.7 shows the results of the initial PlanetLab testbed for the period from from 13^{th} of June 2010 until 20^{th} of June of the same year, where 500 highly connected nodes reproduce the best results that have been achieved previously in simulations only. The top figure shows the RALP values from the Hierarchical Vivaldi and Vivaldi algorithms. Clearly, Vivaldi 2D+H has not yet stabilized and is highly erroneous. Vivaldi 5D shows good behavior at a value of 0.1. However, Hierarchical Vivaldi reaches a near to perfect value of 0.03, similar to the simulations from the previous chapters. Thus, the algorithm has been able to insert nearly all 32 nodes at their correct position using their network coordinates prediction together with the novel error prediction facility. RALP is an application specific measure and translates to the benefit for the peer selection process. The relative error of the well determined group, which ranges from 0.16 to 0.18 is 3 times lower than 5D Vivaldi's (0.50) as well. The lowest figure shows a complete analysis of the relative error. The figure illustrates that Vivaldi 5D's median relative error is at the 90% percentile of the Hierarchical Vivaldi variants.

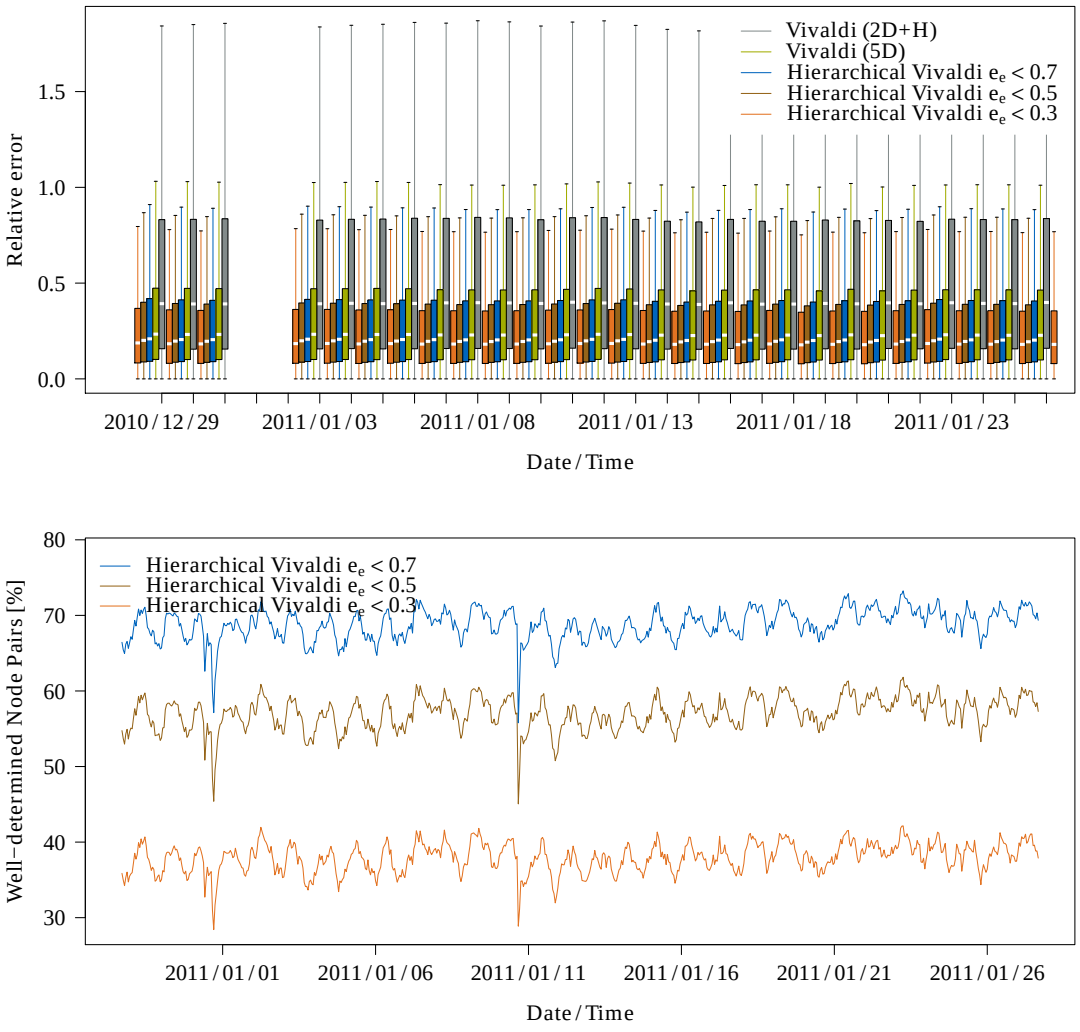

Figure 8.6.: Relative error boxplots (top) and group size (bottom) during final experiment month.

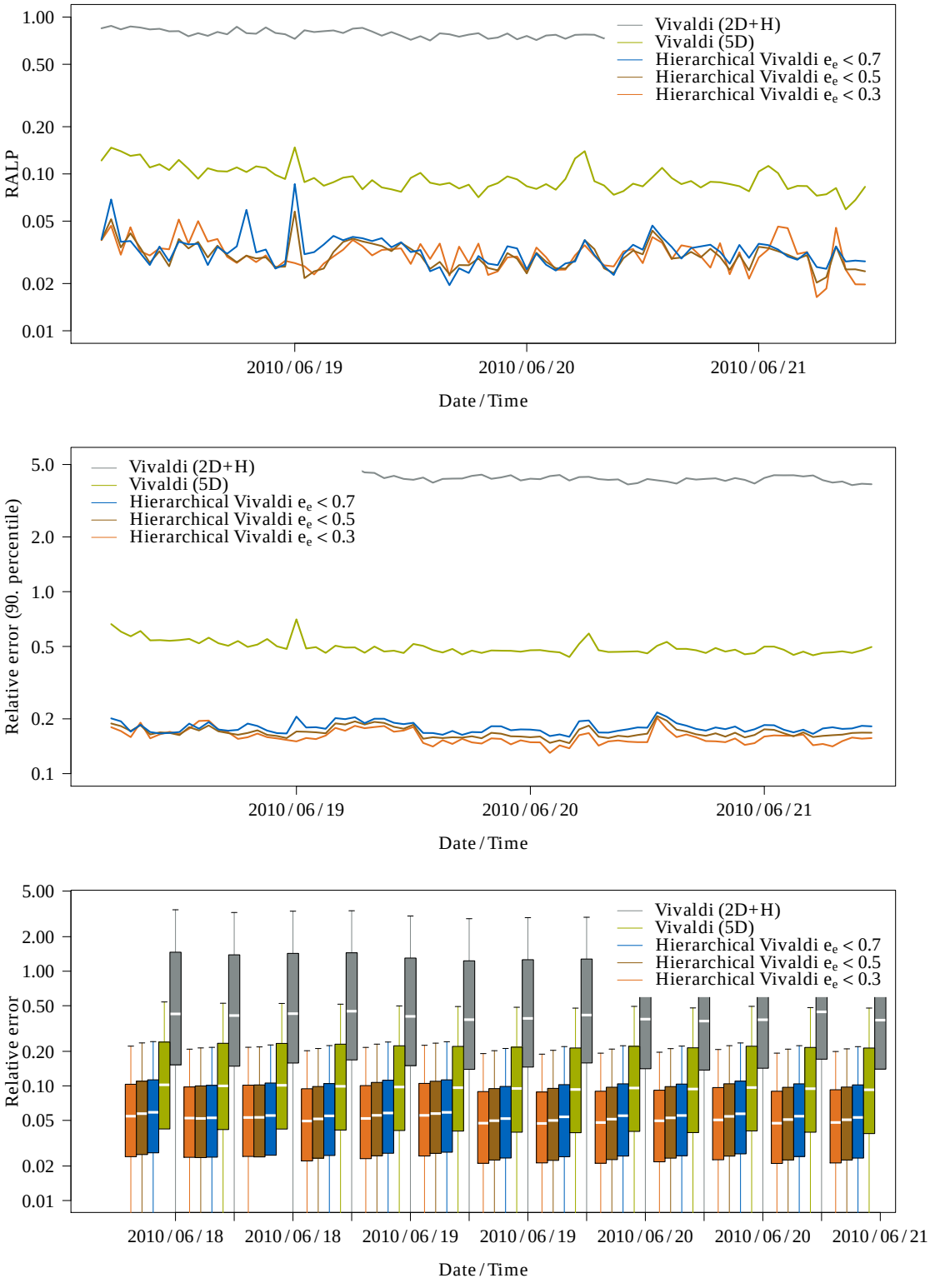

Figure 8.7.: Hierarchical Vivaldi performance during the first experiment phase. 18^{th} to 21^{th} June 2010.

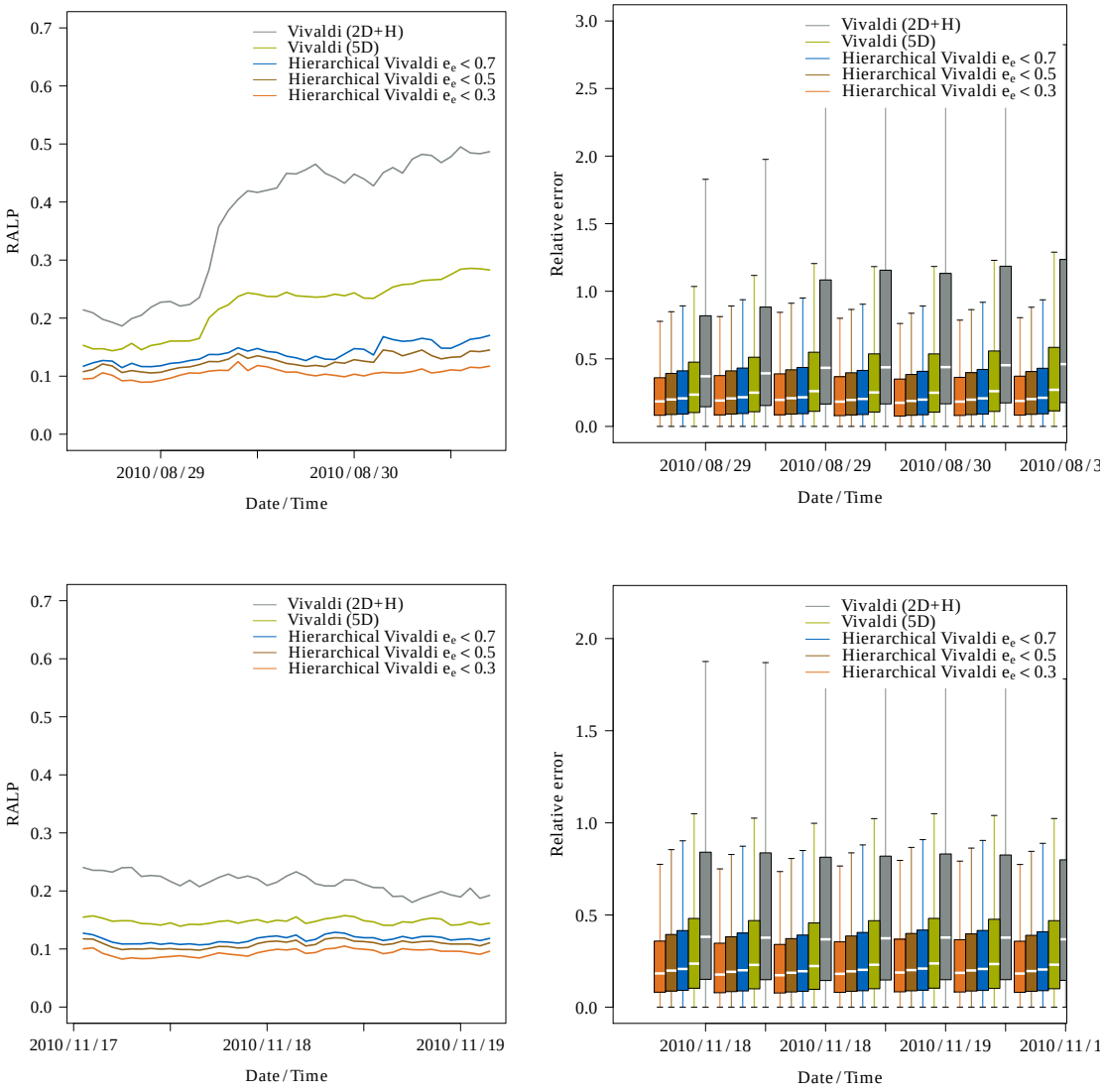

Figure 8.8.: From August 29^{th} until November 18^{th} all PlanetLab nodes were included into the experiment.

The impact of PlanetLab

Stability is a critical feature for network coordinate systems. In a previous chapter I measured stability of simulated coordinates using a change rate. During the Vuze experiment I was able to measure the stability of a large scale system using the relative error and RALP measures. Although only a small fraction of the seven million installations is online at each point in time, the Vuze network is without doubt very large at any point of time during the experiment. Confronting that network with only 500 nodes from the PlanetLab testbed should not have a sizable effect on that network. To investigate that question, the PlanetLab nodes that were previously used for testing and bootstrapping purposes, were added to the active contributor list on October 28^{th}. The results in Figure 8.8 (top row) illustrate the impact of that addition. Both Vivaldi set-ups experience disruptions higher than normal churn. Especially the 2D+Height variant faces a drastic loss of accuracy. Hierarchical Vivaldi exhibit only a moderate reaction both when using RALP or the relative error as measure. Vivaldi 5D increases in the RALP measure from 0.96 to 1.2, which is a modest reaction. However it doesn't reach a RALP measure below 0.97 until the 3^{rd} of October. The stability of Hierarchical Vivaldi is unmatched during the whole experiment.

On November 18^{th} all PlanetLab nodes were shut down. The impact of that action is not reflected in the data (cf. Figure 8.8). It is reasonable, to conclude that that group of nodes was not needed for the stabilization of the network, despite the constant availability and well connected nature of these nodes. In contrast, the joining of a larger group of nodes has a negative effect, once again despite their relative small size compared to the rest of the network. The comparison with Lorch et al. [AL09] concludes favorable. While they use GPS coordinates to produce good initial results, this setup reuses roughly one months old coordinates from the end of the previous PlanetLab activity. Although old coordinates provoke strong reactions in some Vivaldi settings, the network as a whole adapts quickly to the disruption. When it took month in the very first testing phase for all coordinates to adapt, the system here quickly returns to a normal error rate. Finally, I conclude that Hierarchical Vivaldi is much more stable than its non-hierarchical predecessors, which I consider an important benefit.

Htrae [AL09] claims to be superior for their GPS based initial seed of coordinates. From the activation of the PlanetLab nodes, I learnt that using previously used coordinates cuts the initial stabilisation period significantly until a reasonable accuracy is achieved. However, whether any of the Vivaldi solutions or Htrae finds the global optimal solution is neither certain for the present Vuze based evaluation or their XBox network. Both testbeds lack a complete latency matrix that could assess the one or other claim.

Conclusion

I draw several conclusions from the results of this large scale experiment:

First, considering the performance of Hierarchical Vivaldi, the results were superior and met the expectations drawn from the simulation study. Throughout the entire time period Hierarchical Vivaldi delivered results twice as good, during the PlanetLab-only phase even three times as good results in comparison to the same non-hierarchical configuration. Hierarchical Vivaldi is therefore an alternative for systems with highest demands on prediction quality.

Second, as a matter of fact, the results also prove the most basic property of Hierarchical Vivaldi, the separation of nodes into well determined and vaguely determined groups. During the entire period not one of the non-hierarchical configurations, or even any of the well determined groups with a less picky threshold e_e performed better than a pickier configuration.

Third, my proposal proved to be more stable facing churn or massive joining of vaguely determined nodes. While all non-hierarchical configurations got distracted by a moderate join of 500 nodes the hierarchical variant detected those nodes as vaguely determined and was able to deliver stable results.

Considering the non-hierarchical variants, there are also several observations. Especially Vivaldi 5D performed surprisingly good. The reason that performance could not be assessed in simulations is the difference between the time period of the simulation and the experiment. From the previous section we learnt, that it took Vivaldi 5D two months to recover from the re-activation of PlanetLab nodes, therefore the better performance "in the wild" is obvious, because both time periods differ by several orders of magnitude.

The performance of the 2D+H Vivaldi was mediocre, however, considering the extremely low number of dimensions the results are reputable. Vivaldi 2D+H is unable to tolerate nodes with coordinates that do not reflect an accurate position in the network. The built-in error parameters of Vivaldi, that are related to the relative error are not able to distinguish erroneous and well determined nodes. That effect is especially drastic, if only two dimensions express the displacement. Even Vivaldi 5D suffers from that effect judging from the same PlanetLab activation I described previously. However, there may be situations where mediocre stability is tolerable, given the low overhead of three dimensions.

To sum it up, all results are encouraging. The doubts in Vivaldi's abilities to resolve the structure of the underlying network better than at continental level deserve a re-examination. This will happen in the next section.

8.2.2. Performance of Vivaldi-like systems

Analyzing the collected data showed that network coordinates can indeed reveal Internet structures on a regional level.

Figure 8.9 shows the peers in U.S. university networks as seen from university-based reporters at Santa Barbara, Urbana, and Chicago. The box plots show the distribution of the measured and estimated RTTs for the states from west to east. The gray scale indicates the number of reports, i. e. a black box is based on more reports than a light gray box.

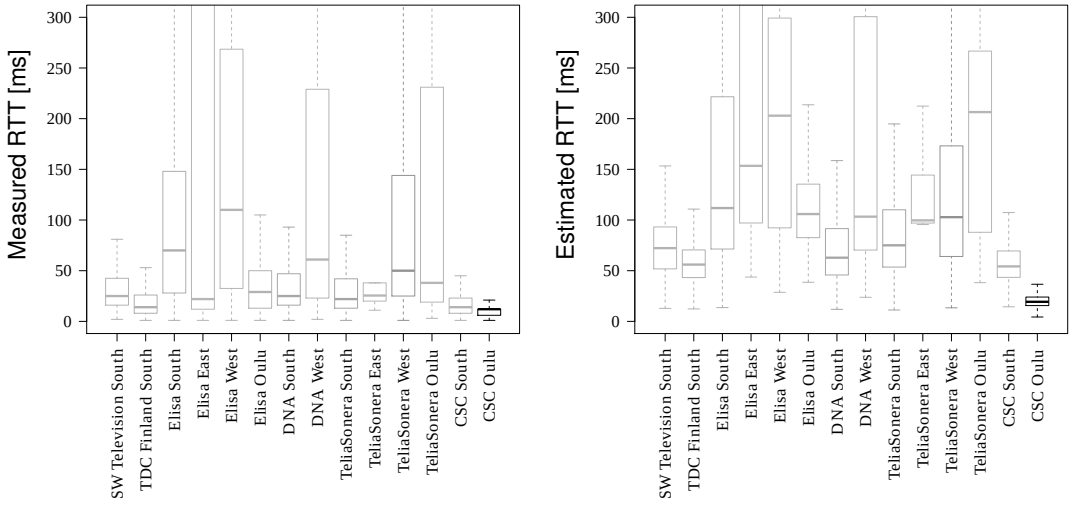

Figure 8.10.: Measured and estimated RTT from Oulu, Finland.

As one can see, the estimated RTT matches the measured RTT, even though the estimated values are higher in general. Still, the estimation indicates the correct trend: For example, peers in the East North Central region yield the best RTTs for the reporters in Urbana and Chicago. The Californian peer is located at the most western position, hence experiences rising RTTs to every other peer. The Vivaldi coordinates reflect this accurately.

The situation is more difficult outside university networks. Figure 8.10, for example, shows the peers in Finland as seen from a reporter in Oulu. Clearly the peers in the CSC network at Oulu are the most preferable. The peers in the TDC, DNA, and CSC networks in South Finland are also acceptable. The peers in the Elisa networks and those in the Telia network at Oulu should better be avoided. In general, the RTT estimation recommends good peers and avoids bad peers.

Typically, neither the autonomous system nor the geo-location are reliable indicators for a peer's performance. Figure 8.11 along with Table 8.2 shows the situation for a reporter in the Kabel BW network in Deißlingen, Germany: Measurement and

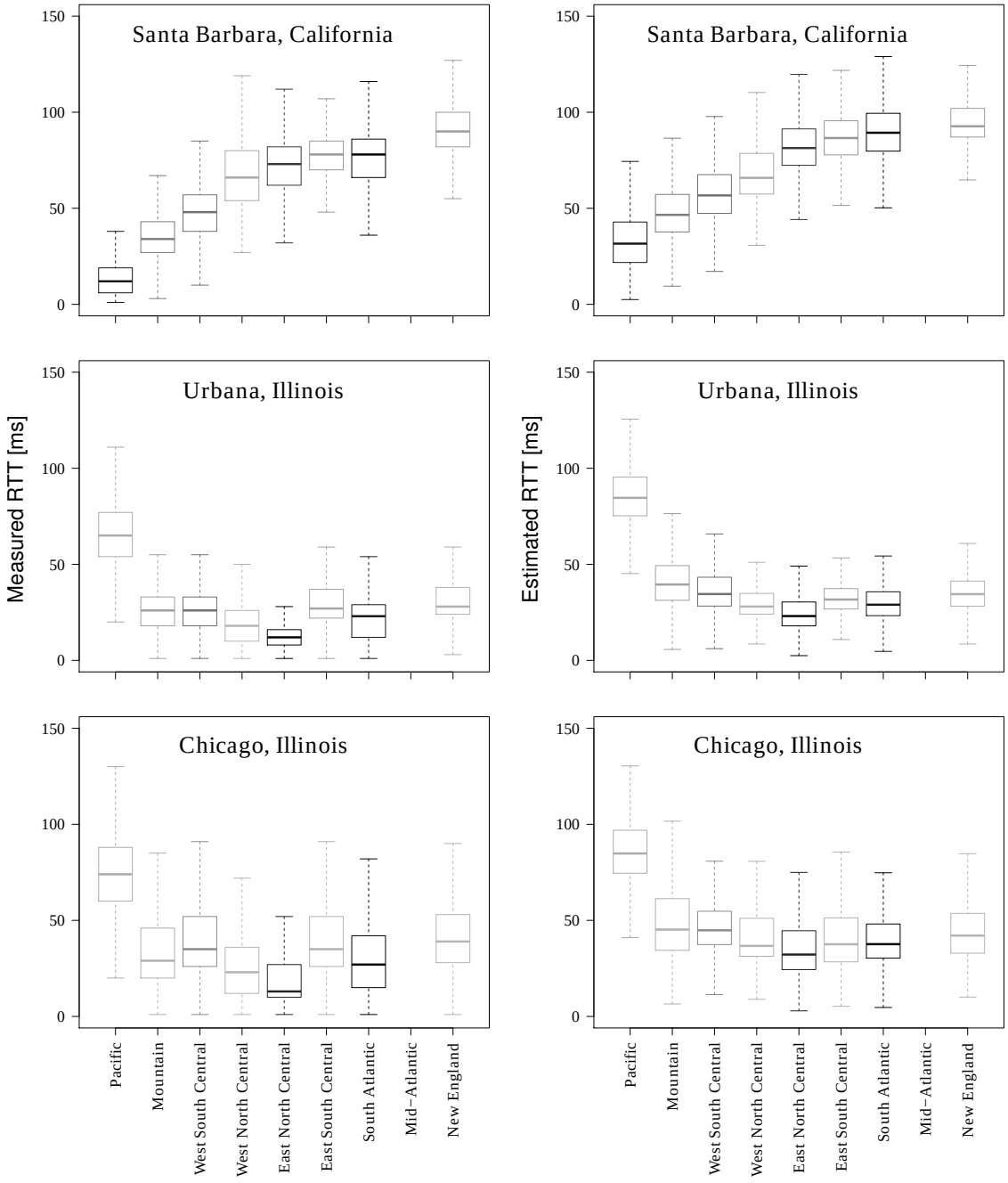

Figure 8.9.: Measured and estimated RTTs in US university networks

Country	Measured RTT [ms]	Estimated RTT [ms]
Germany	45	49
Switzerland	47	80
Poland	55	51
Netherlands	60	61
Austria	65	77
Finland	66	57
Bulgaria	69	69
France	80	101
UK	103	117
Russian Fed.	110	115
USA	184	199
Japan	314	318
India	342	362
Singapore	398	373
Australia	406	443

Table 8.1.: Average International RTTs as seen from Deißlingen, Germany.

Country	Measured RTT [ms]	Estimated RTT [ms]
Saarland	19	29
Mecklenburg V.	33	40
Hessen	39	48
Niedersachsen	39	46
Thuringen	41	45
Berlin	44	43
Bayern	46	47
Sachsen	62	71
Baden W.	64	82
Hamburg	67	78
NRW.	70	92
Bremen	73	94
Sachsen Anhalt	77	97
Schleswig Hol.	78	88
Brandenburg	82	88

Table 8.2.: Average national RTTs as seen from Deißlingen, Germany.

estimation match quite well, but there is no clear correlation between RTT and region or autonomous system. Even more interesting is the consultation of that node's international measurements in Table 8.1. While both on a national and international scale regions are well separated, there is an overlap between regions of Germany and neighbor states for the European Union. Not only is Switzerland closer to Baden Württemberg than Bavaria, even Amsterdam seems closer, compared to Nordrhein Westfalen. For reasons of brevity a more complete version of the international and national table is in the appendix, along with the number of clients per country (Table A.1). University networks are different. They generally exhibit a good correlation between location and RTT, a fact well known in literature [SSW10b].

Using universities as example serves well for showing the limits of a network coordinate algorithm. In Figure 8.12 Californian Universities and their respective latencies (left) measured from Santa Barbara. The figure is sorted from north to south. All median RTT values settle at 11 milliseconds, however the variance of the RTT is high. In that situation the algorithm is unable to come up with a prediction: The median values of the prediction are three times higher than their actual counterparts. I did a further investigation, to show that that performance is really an outcome of the variance in the RTT values. Figure B.1 shows the same data, broken down at a month scale. It was moved to the appendix for reasons of brevity. Clearly the prediction in January is just as expected, judging from the RTT values on the left to the estimated values on the

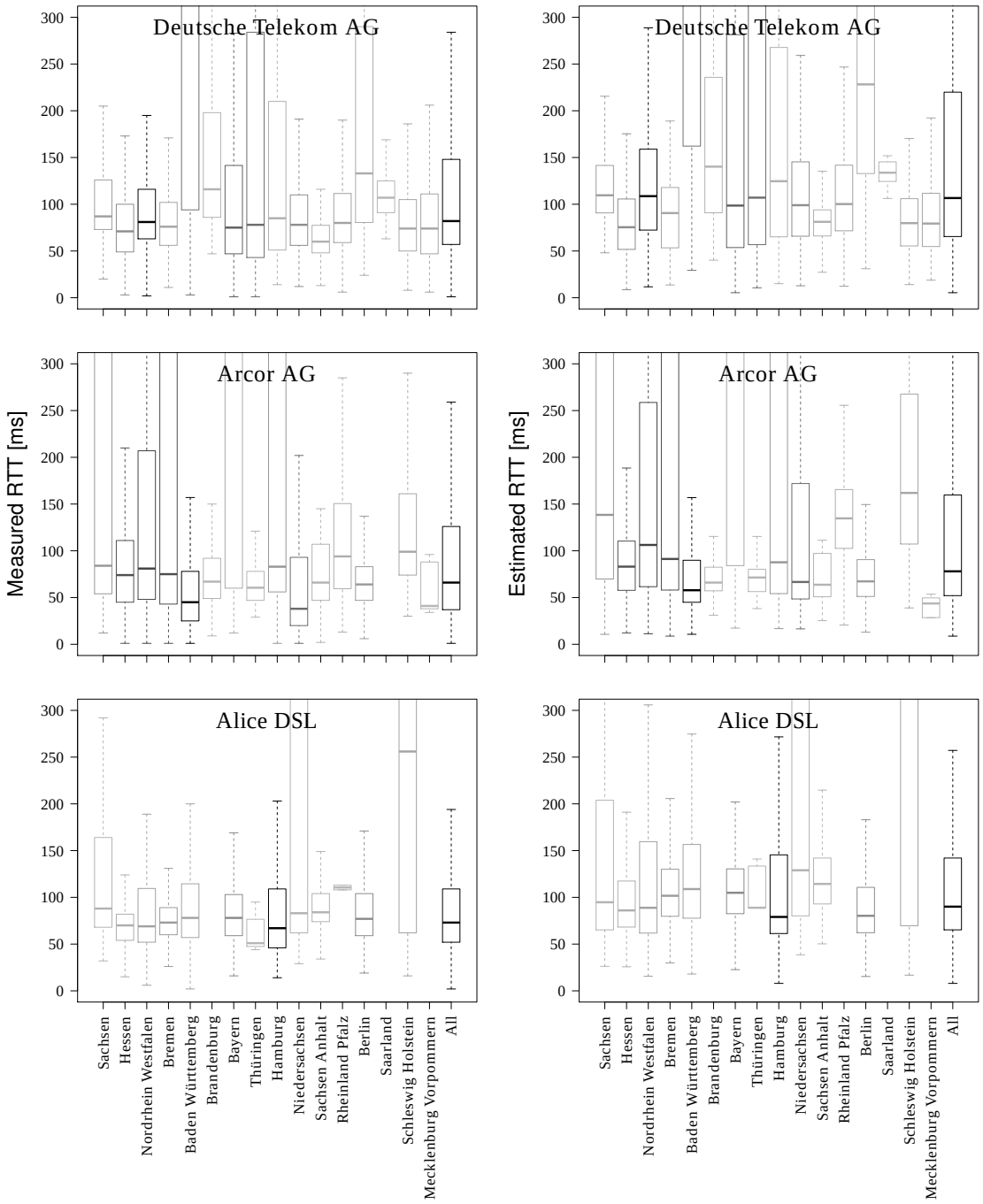

Figure 8.11.: Regional comparison of autonomous systems in Germany. Measurement from Baden Würtemberg, Kabel BW.

right. However the other month lack measurements, or exhibit very imprecise latency behavior, which the algorithm is clearly unable to cope with. This is one of the rare cases, where an academic network exhibits high instabilities. Hence although it seems, that Vivaldi failed to come up with a decent prediction, that figure clearly shows, that with decent input data, Vivaldi is able to deliver such accuracy.

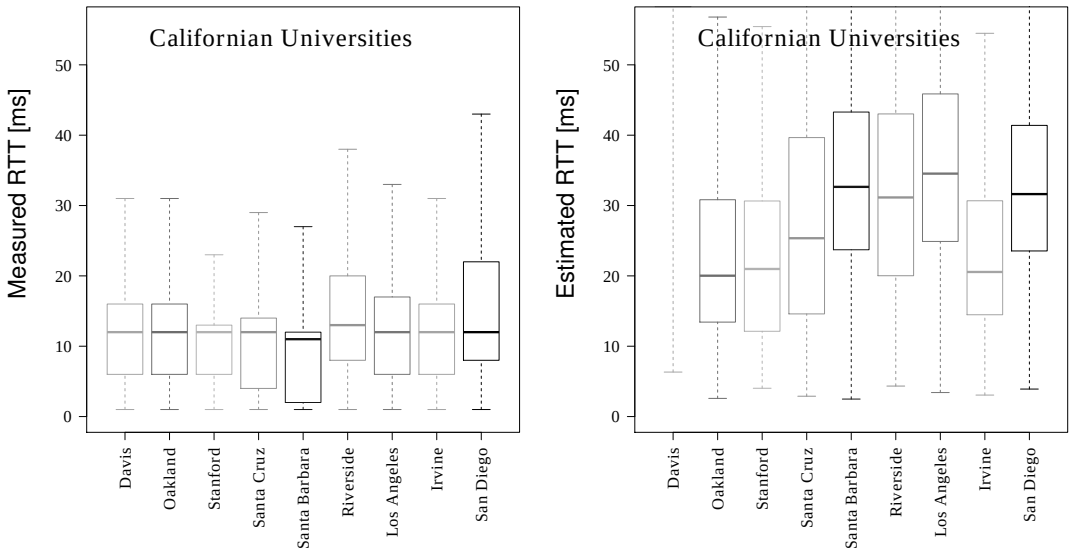

Figure 8.12.: Measured and estimated RTT between Californian Universities from north to south.

When separating on the AS level, one expects, that traffic under provider control will be prioritised among other traffic. This is confirmed by Figure 8.13 on page 141. Each of the first three boxplots represent the RTT of the biggest ASes measured from that node. The box labeled "Kabel Deutschland" is the node's provider. However, although the median values of Kabel Deutschland are in the lowest range of that group, it has exhibits a remarkably high variance, which is unexpected. The effect is especially drastic, as Kabel Deutschland is a big national provider in Germany. In the light of Google's Glasnost project, which tested residential Internet connections for signs of P2P traffic shaping [MA11], the figure deserves attention. The Glasnost tool and respective publication accused Kabel Deutschland of traffic shaping. And indeed a quick comparison with Figure 8.14 reveals higher variance in the Kabel Deutschland network, compared to the Kabel BW network, which although the name might suggest it, is not affiliated with Kabel Deutschland. However that second data comes from a peer in Baden Württemberg, unfortunately there was no second reporting peer from Mecklenburg Vorpommern.

The most important property of network coordinate algorithms is their ability to recommend for or against a peer. Table 8.3 shows this feature in detail. It contains the 18 most active reporters. Assume a peer probes one after another the candidates from

	Best	Best 3	3 of 5	Best 6	6 of 9	Best 9
Stevenage, UK	1	5	5	20	20	133
Kirkland, WA	2	11	11	68	12	68
Arlington, TX	2	12	7	44	12	44
Chicago, IL	2	13	6	40	13	52
Oak Creek, WI	4	80	60	80	74	93
Meno. Falls, WI	5	13	8	31	23	69
Sofia, Bulgaria	6	6	5	17	6	305
Deissling, Germany	6	12	10	30	12	121
Santa Barbara, CA	7	11	7	26	11	26
Rostock, Germany	9	129	38	129	38	129
Oulu, Finland	17	20	17	35	17	35
Vienna, Austria	31	31	15	73	30	73
Austin, TX	36	36	27	188	36	188
Saint Austell, UK	120	120	38	120	72	329
Almere, Netherlands	185	213	185	225	185	295
Geelong, Australia	213	213	107	213	126	318
Urbana, IL	214	285	176	285	179	332
Odessa, TX	469	469	88	469	101	471

Table 8.3.: Number of probes to find the RTT-wise best peers.

a sorted recommendation list, which is created by ranking the peers according to their estimated RTT. The table gives the number of candidates that the peer must probe to find the best, the three best, three out of the five best, etc. For more than half of the listed nodes, less than ten probes suffice to find the globally best peer; and about 20 probes suffice to find six of the nine best peers. However there are situations, where a misidentification of the best peer leads to an unacceptable high number of measurements. For example the peer in Odessa, Texas needs 469 probes, to identify it's closest peer. However, this does not reliably forecast its overall performance. The third best peer is identified as second best peer, hence is found in two probes, as the following table excerpt (sorted by estimated latency) shows.

Position	ID	Estimated RTT	Measured RTT	
1	88494483e2cb5d431	81	112	
2	5b0e00bc3b29d550a	99	92	*(third best)*
3	eb7e10440c1594239	100	105	
	...			
469	852cc323d67d61663	172	87	*(best peer)*

As a focus on the exact list of the n best peers is not necessarily the best measure, we introduced the candidate rank in [EF11]. Hence Figure 8.15 shows that average RTT overhead of a peer when it connects to the k best ranked candidates. Even though

the reporter at Urbana finds its globally optimal peer only at the 214th rank in the recommendation list (cf. Table 8.3), it will have only a negligible average RTT overhead of less than 5 milliseconds when it just connects to the five best ranked candidates. In general, the overhead will be 5 - 15%.

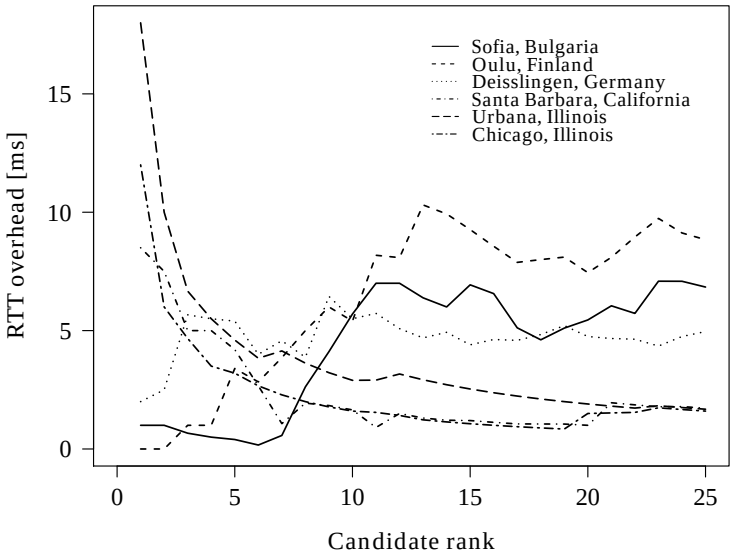

Figure 8.15.: Absolute RTT penalty when using the k best-ranked peers.

Figure 8.16 shows how many of the k best ranked candidates are located in the same region or AS as the peer. The figure shows the average of the 18 most active reporters. Clearly a network coordinate algorithm is able to create locality in more than 60% of all settings.

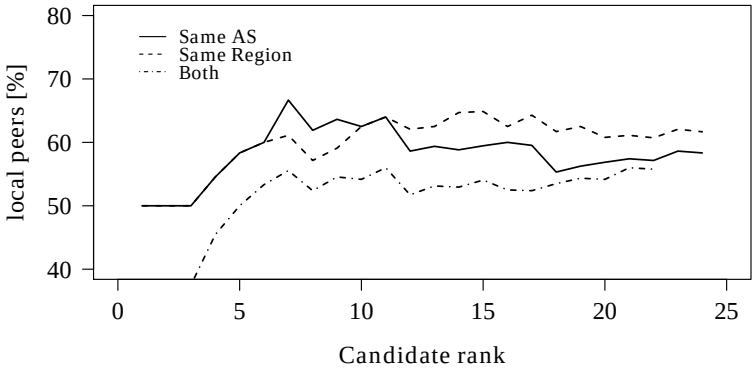

Figure 8.16.: Cumulative fraction of peers in the same region or AS.

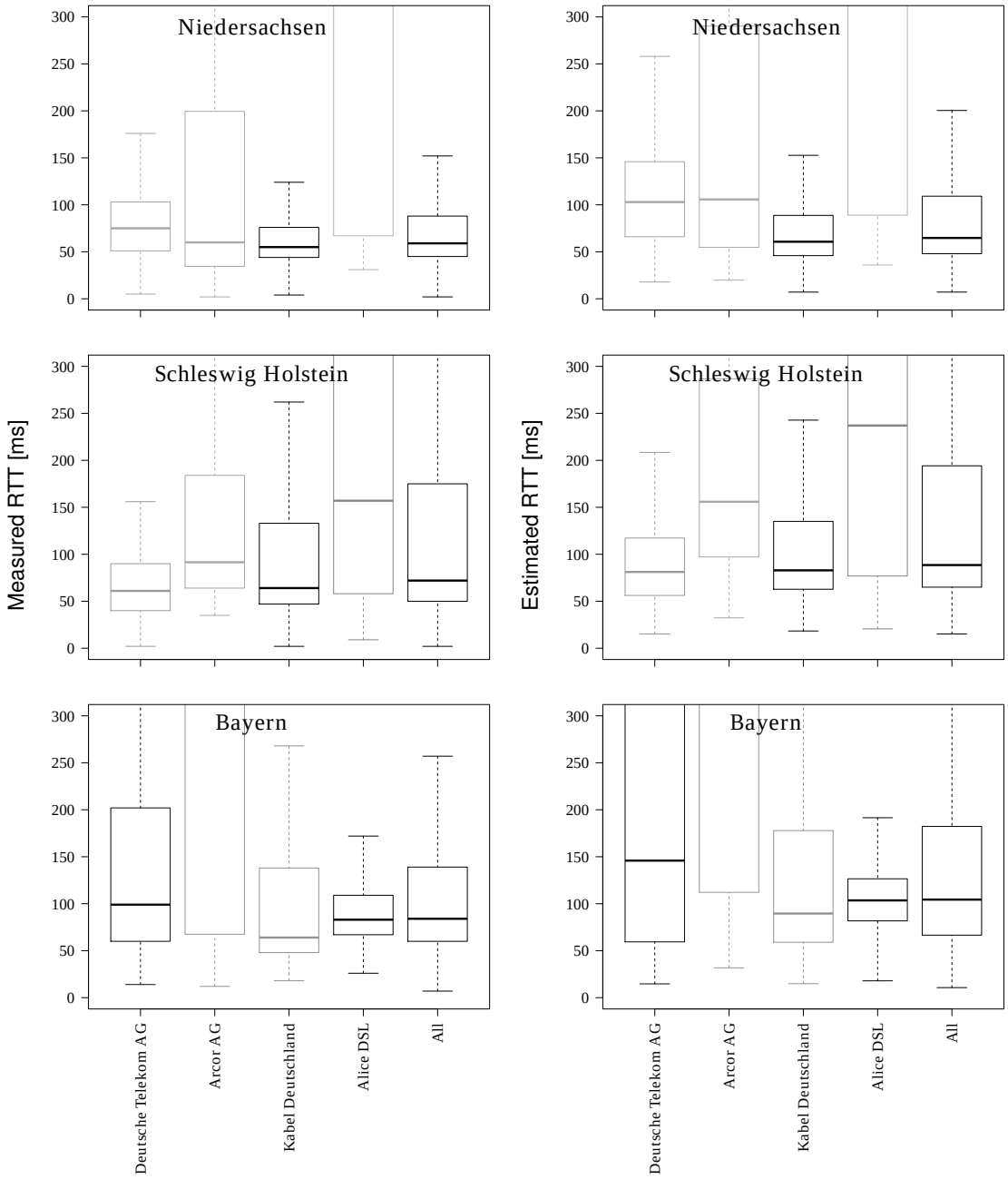

Figure 8.13.: Comparison of autonomous systems at regional scale. Measurement from Mecklenburg Vorpommern, Kabel Deutschland

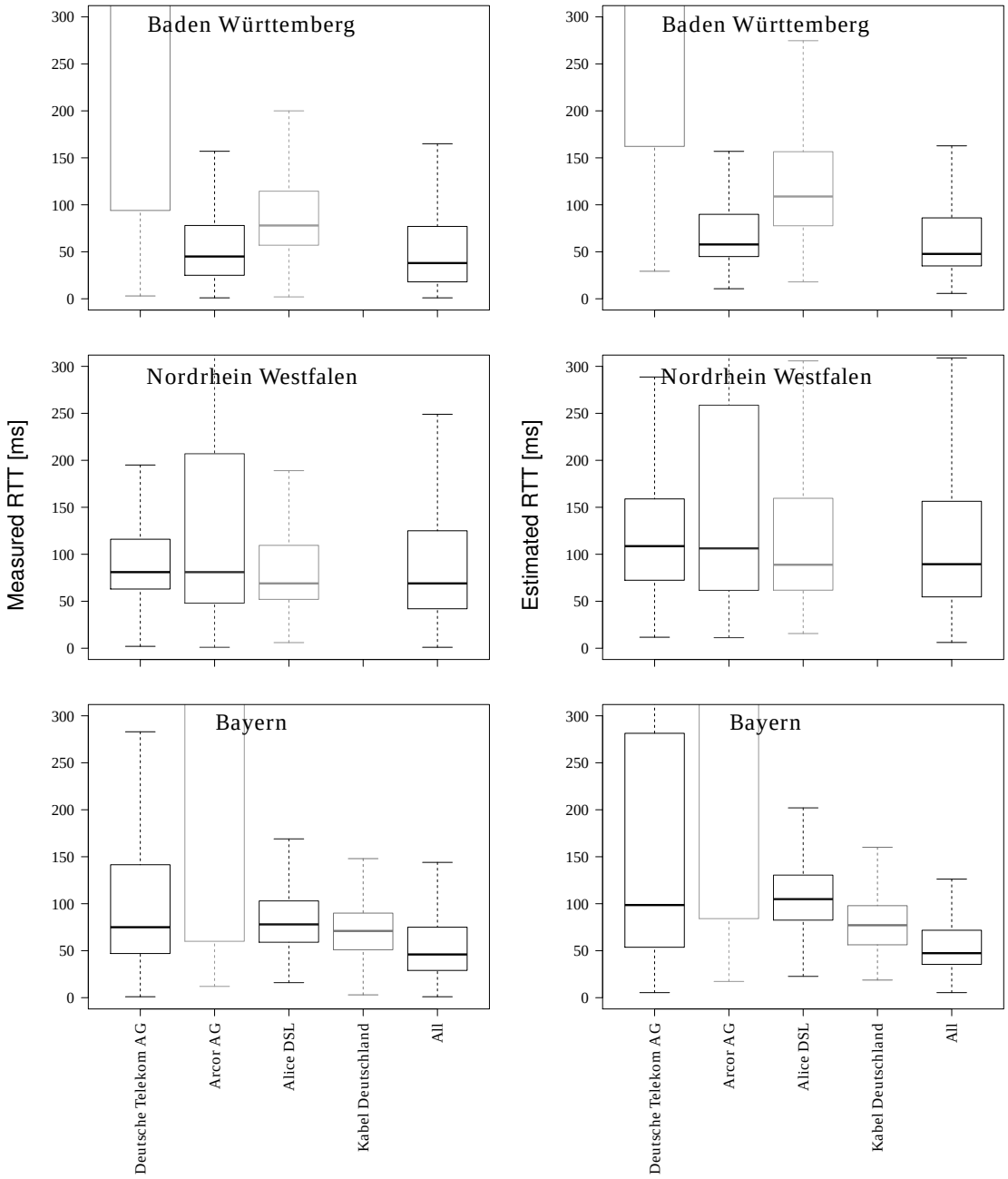

Figure 8.14.: Comparison of autonomous systems at regional scale. Measurement from Baden Württemberg, Kabel BW

9. Conclusion

In this thesis, I have presented Hierarchical Vivaldi, a new algorithm that supports the selection of peers in proximity aware decentral peer-to-peer overlay networks. The algorithm is based on the widely used Vivaldi algorithm. It optimizes Vivaldi's peer selection process with an error estimation, which is derived from a novel concept of using a hierarchical embedding of measured latencies and their respective embedding errors. The algorithm's fundamental contribution is that it considers only those peers whose estimated prediction error is below a given threshold. The resulting accuracy outperforms the non-hierarchical approaches by an order of magnitude.

I presented an in-depth analysis of Internet latency prediction based on the Vivaldi and Pyxida algorithm. In a simulation study, I analyzed the influence of various parameter combinations and several sets of Internet traffic traces on these algorithms. To conduct the study, I created a simulator using the components of the IGOR ecosystem. From the results, I found that most of the parameters that were recommended in the original papers are a magnitude too high. They adapt quickly to changes in the network and therefore issue accurate predictions early. But by choosing more conservative parameters coordinates adapt fast and do not exhibit huge instabilities as a response to disruptions in latencies. Therefore I recommended modified parameters that improve the algorithms' performance significantly.

Using the parameters deduced from my study I conducted a simulation study on my Hierarchical Vivaldi algorithm. The results were more accurate than the non-hierarchical approaches by an order of magnitude. Even when I compared a 22 dimensional Vivaldi to a 5 dimensional Hierarchical Vivaldi (in the top space, 3 dimensions in the lower spaces), the results of my proposal were still more accurate. By including the algorithm in the popular Vuze client, I reassured the results from the simulation study based on the performance of 7 million actual installations. Clearly Hierarchical Vivaldi is more accurate than Vivaldi, and outperforms Vivaldi during the whole measurement. Because it identifies peers with high errors, it is more robust to a huge amount of peers joining the system with vaguely determined coordinates.

Finally, I re-evaluated the popular claim, that Vivaldi based network coordinate systems can only resolve the structure of the underlying network to a continental level. Using the massive amount of measurement data from the evaluation study, I was able to invalidate that claim. The performance is much better than popular believed. Predictions that resolve structures to a regional level are possible. For example, as demonstrated one can clearly resolve the distances in the US from east to west. I also explored the

limitations of the algorithm. For example I tried to resolve Californian Universities from north to south. These results were imprecise. Hence I evaluated smaller time frames of the same data. That revealed that the real culprit were large periods of flaws in the input RTT data, which was reflected in the prediction. Hence another contribution of my thesis is the retrieval of the network coordinates' honor, that network coordinates were prematurely dismissed as not being helpful in the construction of peer-to-peer overlay networks. It demonstrates that Vivaldi-like network coordinates can indeed successfully recommend beneficial peers, even though the application layer round trip times of many peers vary largely.

Thus I conclude that the goals of this thesis have been met: The primary goal was to improve the overlay's awareness of its underlay. Hierarchical Vivaldi improves the prediction quality of the Vivaldi algorithm by an order of magnitude. During the development of that algorithm I analyzed the existing algorithms in-depth. All simulations were carried out using components of the IGOR ecosystem, which I extended to include simulation capabilities. Another part of the ecosystem is the application level library, that supports complex endpoint operations. With that ecosystem I provide a toolset to developers to foster the creation of P2P applications, which was also a goal of my thesis.

Future Work

Even though my thesis thoroughly studied Vivaldi and Hierarchical Vivaldi, further analysis and improvements of Hierarchical Vivaldi are possible. First of all the hierarchical extension can be deeper integrated into the Vivaldi algorithm. Currently Vivaldi uses its local and the remote error to compute the damping factor δ, which controls the actual adjustment of the node's coordinates. Generally, I found that the error estimator e_e is much more accurate than Vivaldi's built-in local error. In all simulations e_e was able to effectively separate the well determined nodes from the vaguely determined ones. Hence computing the damping factor δ depending on e_e will lead to a coordinate adjustment in a well determined amount.

Several publications advocate different spaces for the Vivaldi embedding. The hyperbolic space is evaluated in [LS08] with mixed results. The authors propose a heuristic ThresholdHyperbolic, that uses hyperbolic coordinates for latency close and Euclidean coordinates for distant peers. Under the assumption that the relative error that is embedded in spaces $\wp_0$ is significant smaller than the actual RTT using a hyperbolic embedding in the spaces below $\wp_0$ might yield an improved performance or reduce the resource demand of Hierarchical Vivaldi. An improvement in prediction quality could allow to use $\mathrm{T}_{max} = 1$, which reduces the overhead to only two additional spaces.

A. Extended Tables

Country	Clients	Measured RTT [ms]	Estimted ETT [ms]
Germany	4060	45	49
Switzerland	227	47	80
Poland	619	55	51
Hungary	402	56	60
Belgium	629	58	61
Netherlands	406	60	61
Austria	362	65	77
Finland	164	66	57
Denmark	125	68	67
Bulgaria	74	69	69
Romania	371	71	68
Sweden	456	77	78
Italy	2074	79	86
France	1128	80	101
Spain	633	93	102
UK	3763	103	117
Russian Fed.	993	110	115
Greece	648	112	141
Portugal	558	118	139
Canada	1298	178	194
USA	6545	184	199
Turkey	218	222	288
Malaysia	327	274	361
Mexico	241	302	366
Brazil	576	308	357
Japan	236	314	318
India	3780	342	362
South Africa	287	373	516
Singapore	248	398	373
Australia	1134	406	443

Table A.1.: Average International RTTs as seen from Deißlingen, Germany (extended version).

Country	Clients	Measured RTT [ms]	Estimted ETT [ms]
Saarland	37	19	29
Mecklenburg V.	55	33	40
Hessen	255	39	48
Niedersachsen	212	39	46
Thuringen	132	41	45
Berlin	280	44	43
Bayern	381	46	47
Sachsen	182	62	71
Baden W.	430	64	82
Hamburg	309	67	78
NRW.	1036	70	92
Bremen	159	73	94
Sachsen Anhalt	58	77	97
Schleswig Hol.	180	78	88
Brandenburg	30	82	88

Table A.2.: Average national RTTs as seen from Deißlingen, Germany (extended version).

B. Extended Figures

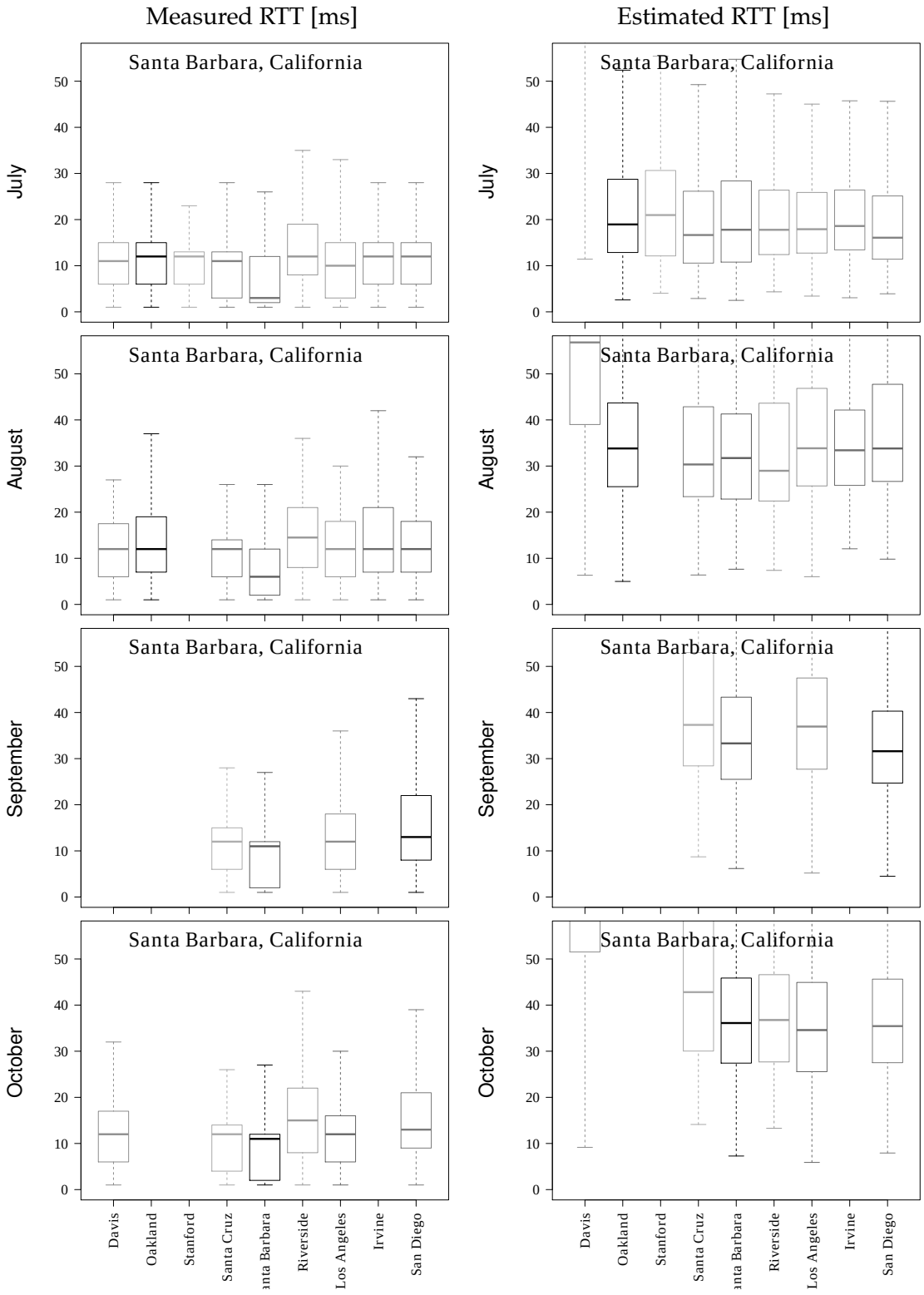

Figure B.1.: Californian universities from North to South. Timeline of measured (left) and estimated (right) RTTs.

List of Tables

List of Algorithms

List of Figures

Bibliography

[AEHF08] Bernhard Amann, Benedikt Elser, Yaser Houri, and Thomas Fuhrmann. Igorfs: A distributed p2p file system. In *Proceedings of the P2P'08: Proceedings of the 8th IEEE International Conference on Peer to Peer Computing*, Aachen, Germany, September 8 – 11, 2008. IEEE Computer Society.

[AFS07] Vinay Aggarwal, Anja Feldmann, and Christian Scheideler. Can ISPS and P2P users cooperate for improved performance? *ACM SIGCOMM Computer Communication Review*, 37(3):29–40, 2007.

[AKR+05] M. Adler, R. Kumar, K. Ross, D. Rubenstein, T. Suel, and D.D. Yao. Optimal peer selection for p2p downloading and streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1538 – 1549 vol. 3, March 2005.

[AL09] Sharad Agarwal and Jacob R. Lorch. Matchmaking for online games and other latency-sensitive p2p systems. *ACM SIGCOMM Computer Communication Review*, 39(4):315–326, 2009.

[AZ] PlanetLab-to-Azureus Latency Trace in Matrix Form (not augmented), `http://www.eecs.harvard.edu/~syrah/nc`, accessed 19. November 2011.

[BBB11] Steven Bauer, Robert Beverly, and Arthur Berger. Measuring the state of ecn readiness in servers, clients, and routers. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement*, IMC '11, Berlin, GermanACMy, November 2011. ACM.

[Bel10] David Belson. Akamai state of the internet report, q4 2009. *SIGOPS Oper. Syst. Rev.*, 44:27–37, August 2010.

[Ber] Thomas Bernard. Miniupnp project homepage. `http://miniupnp.free.fr/`. Accessed 12.04.2010.

[BND10] A. Bulkowski, E. Nawarecki, and A. Duda. Indexing and searching learning objects in a peer-to-peer network. In *Education Engineering (EDUCON), 2010 IEEE*, pages 1665 –1670, April 2010.

[CB08] David R. Choffnes and Fabián E. Bustamante. Taming the torrent: A practical approach to reducing cross-ISP traffic in peer-to-peer systems. *ACM SIGCOMM Computer Communication Review*, 38(4):363–374, 2008.

[CB09] David R. Choffnes and Fabian B. Bustamante. What's Wrong with Network Positioning and Where Do We Go From Here? Technical Report NW-EECS-09-03, North Western University, 2009.

[CB10] David R. Choffnes and Fabian E. Bustamante. Pitfalls for testbed evaluations of internet systems. *ACM SIGCOMM Computer Communication Review*, 40:43–50, April 2010.

[CCL+04] Rui Castro, Mark Coates, Gang Liang, Robert Nowak, and Bin Yu. Network tomography: recent developments. *Statistical Science*, 19:499–517, 2004.

[CDK+03] Russ Cox, Frank Dabek, Frans Kaashoek, Jinyang Li, and Robert Morris. Practical, distributed network coordinates. In *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, Cambridge, Massachusetts, November 2003. ACM SIGCOMM.

[CDS74] V. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675, December 1974.

[Che96] Stuart Cheshire. Latency and the quest for interactivity. White paper, University of Stanford, Novemver 1996.

[CHM+02] Ian Clarke, Theodore W. Hong, Scott G. Miller, Oskar Sandberg, and Brandon Wiley. Protecting Free Expression Online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.

[CHNY02] Mark Coates, Alfred Hero, Robert Nowak, and Bin Yu. Internet tomography. *IEEE Signal Processing Magazine*, 19:47–65, 2002.

[Cis10] Cisco Systems. Cisco Visual Networking Index: Forecast and Methodology, 2009-2014, June 2010.

[CKV10] L. Cavedon, C. Kruegel, and G. Vigna. Are BGP Routers Open To Attack? An Experiment. In *Proceedings of the iNetSec Conference*, Sophia, Bulgaria, March 2010.

[CLL+99] James Cowie, Hongbo Liu, Jason Liu, David Nicol, and Andy Ogielski. Towards realistic million-node internet simulations. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, June 1999.

[CLY+09] Rubén Cuevas, Nikolaos Laoutaris, Xiaoyuan Yang, Georgos Siganos, and Pablo Rodriguez. Deep diving into bittorrent locality. In *Proceedings of the 5th international student workshop on Emerging networking experiments and technologies*, Co-Next Student Workshop '09, pages 7–8, New York, NY, USA, 2009. ACM.

[Coh03] Bram Cohen. Incentives Build Robustness in BitTorrent. In *Proc. of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, USA, May 2003.

[Com] OMNeT++ Community. Omnet++ network simulation framework. `http://www.omnetpp.org/`, accessed 19. November 2011.

[Cro07] Jon Crowcroft. Net neutrality: the technical side of the debate: a white paper. *ACM SIGCOMM Computer Communication Review*, 37:49–56, January 2007.

[CRR+05] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. A case study in building layered DHT applications. *ACM SIGCOMM Computer Communication Review*, 35(4):108, 2005.

[CvdGLR10] C. Contavalli, W. van der Gaast, S. Leach, and D. Rodden. Client IP information in DNS requests. Internet-Draft draft-vandergaast-edns-client-ip-01, Internet Engineering Task Force, May 2010. Work in progress.

[CvdGLR11] C. Contavalli, W. van der Gaast, S. Leach, and D. Rodden. Client subnet in DNS requests. Internet-Draft draft-vandergaast-edns-client-subnet-00, Internet Engineering Task Force, January 2011. Work in progress.

[CWS+09] Yang Chen, Xiao Wang, Xiaoxiao Song, Eng Keong Lua, Cong Shi, Xiaohan Zhao, Beixing Deng, and Xing Li. Phoenix: Towards an accurate, practical and decentralized network coordinate system. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 313–325, Berlin, Heidelberg, 2009. Springer-Verlag.

[DCKM04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *Proceedings of the ACM SIGCOMM '04 Conference*, Portland, Oregon, August 2004.

[DHGS07] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC'07)*, San Diego, CA, USA, October 2007.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly

available key-value store. *ACM SIGOPS Operating Systems Review*, 41:205–220, 2007.

[DKF08] Pengfei Di, Kendy Kutzner, and Thomas Fuhrmann. Providing KBR service for multiple applications. In *The 7th International Workshop on Peer-to-Peer Systems (IPTPS '08)*, St. Petersburg, U.S., February 2008.

[DKR05] X. Dimitropoulos, D. Krioukov, and G. Riley. Revisiting Internet AS-level Topology Discovery. In *Passive and Active Network Measurement Workshop (PAM)*, pages 177–188, Boston, MA, Mar 2005. PAM 2005. Illustration taken from `http://www.caida.org/research/topology/as_core_network/`.

[DMG+10] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, Krishna P. Gummadi, Ratul Mahajan, and Stefan Saroiu. Glasnost: Enabling End Users to Detect Traffic Differentiation. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[DMMP09] John R. Douceur, James W. Mickens, Thomas Moscibroda, and Debmalya Panigrahi. Thunderdome: discovering upload constraints using decentralized bandwidth tournaments. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 193–204, New York, NY, USA, 2009. ACM.

[DMMP10] J.R. Douceur, J. Mickens, T. Moscibroda, and D. Panigrahi. Collaborative measurements of upload speeds in p2p systems. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –9, March 2010.

[Dou] William L. Dougherty. It's still the latency stupid. `http://www.edgeblog.net/2007/its-still-the-latency-stupid/`.

[DV09] Dominik Vallendor. Umsetzung und Evaluation des Peer-to-Peer-Overlay-Netzes IGOR in der Netzwerk-Simulations-Umgebung SSFNet, 2009. Diplomarbeit, Universität Karlsruhe.

[DZD+03] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a Common API for Structured Peer-to-Peer Overlays. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, 2003.

[EF11] Benedikt Elser and Thomas Fuhrmann. Here is Your Peer! – Locating Peers on a Regional Level with Network Coordinates. In *Proceedings of the P2P'11: Proceedings of the 11th IEEE International Conference on Peer to Peer Computing*, Kyoto, Japan, August 2011.

[EFF09] Benedikt Elser, Andreas Förschler, and Thomas Fuhrmann. Tuning Vivaldi: Achieving Increased Accuracy and Stability. In *Proceedings of the Fourth International Workshop on Self-Organizing Systems*, Zurich, Switzerland, December 2009.

[EFF10] Benedikt Elser, Andreas Förschler, and Thomas Fuhrmann. Hierarchical Vivaldi – Cherry Picking on Network Coordinates. In *Proceedings of the P2P'10: Proceedings of the 10th IEEE International Conference on Peer to Peer Computing*, Delft, The Netherlands, August 2010.

[Els11] Benedikt Elser. Vuze 2010/11 Dataset, 2011. `http://www.so.in.tum.de/~elser/Vuze-Data.html`.

[ES09] J. Brutlag E. Schurman. The user and business impact of server delays, additional bytes, and http chunking in web search, June 2009. Presented at the O'Reilly Web Performance & Operations Converence.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413, August 1993.

[FJJ+01] Paul Francis, Sugih Jamin, Cheng Jin, Yixin Jin, Danny Raz, Yuval Shavitt, and Lixia Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM Transactions on Networking*, 9(5):525–540, 2001.

[Fra11] Benedikt Fraunhofer. Entwicklung eines Azureus-Plugins zur Latenzvorhersage mit hierarchischen Netzwerkkordinaten, 2011. Diplomarbeit, Technische Universität München.

[Fö09] Andreas Förschler. Einbettung von Peer-to-Peer-Netzwerkgraphen zur Vorhersage von Paketumlaufzeiten, 2009. Diplomarbeit, Universität Karlsruhe.

[Get11] Jim Gettys. Bufferbloat: Dark buffers in the internet. *IEEE Internet Computing*, 15:96, 95, 2011.

[GF05] Saikat Guha and Paul Francis. Characterization and measurement of tcp traversal through nats and firewalls. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 18–18, Berkeley, CA, USA, 2005. ACM.

[GMG+04] K.P. Gummadi, H.V. Madhyastha, S.D. Gribble, H.M. Levy, and D. Wetherall. Improving the reliability of internet paths with one-hop source routing. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6*, pages 13–13. USENIX Association, 2004.

[GN11] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. *ACM Queue: Tomorrow's Computing Today*, 9:40:40–40:54, November 2011.

[GSG02] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: estimating latency between arbitrary internet end hosts. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 5–18. ACM, 2002.

[Hud08] Stefan Hudelmaier. Realisierung des Extensible Messaging and Presence Protocol (XMPP) auf Basis eines strukturierten Overlay Netzes, 2008. Diplomarbeit, Technische Universität München.

[HUKC+11] Aymen Hafsaoui, Guillaume Urvoy-Keller, Denis Collange, Matti Siekkinen, and Taoufik En-Najjary. Understanding the impact of the access technology: the case of web search services. In *Proceedings of the Third international conference on Traffic monitoring and analysis*, TMA'11, pages 37–50, Berlin, Heidelberg, 2011. Springer-Verlag.

[Inca] OpenDNS Inc. Global internet speedup. `http://www.afasterinternet.com/`.

[Incb] Vuze Inc. Vuze plug in api. `http://wiki.vuze.com/w/Plugin_Development_Guide`.

[Int11] CBS Interactive. Cnet download.com (website), 2011. `http://download.cnet.com/`.

[Jou] The Wall Street Journal. Rush to fix quake-damaged undersea cables. `http://online.wsj.com/article/SB10001424052748704893604576199952421569210.html`.

[KCF05] Kendy Kutzner, Curt Cramer, and Thomas Fuhrmann. A self-organizing job scheduling algorithm for a distributed vdr. In *Workshop "Peer-to-Peer-Systeme und -Anwendungen", 14. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2005)*, Kaiserslautern, Germany, 2005.

[KF06] Kendy Kutzner and Thomas Fuhrmann. The IGOR file system for efficient data distribution in the GRID. In *Proceedings of the Cracow Grid Workshop CGW 2006*, 2006.

[KMS+09] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize cdn performance. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pages 190–201, New York, NY, USA, 2009. ACM.

[KR04] David R. Karger and Matthias Ruhl. Diminished chord: A protocol for heterogeneous subgroup formation in peer-to-peer networks. In *Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, pages 288–297, San Diego, CA, USA, 2004.

[KR07] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (4th Edition)*. Addison Wesley, 4 edition, April 2007.

[Kut08] Kendy Kutzner. *The Decentralized File System Igor-FS as an Application for Overlay-Networks*. PhD thesis, Universität Karlsruhe (TH), 2008.

[KWNP10] Christian Kreibich, Nicholas Weaver, Boris Nechaev, and Vern Paxson. Netalyzr: illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 246–259, New York, NY, USA, 2010. ACM.

[Lau04] Julian Laub. *Non metric pairwise proximity data*. Dissertation, Technische Universität Berlin, 2004.

[LBL+09] Cristian Lumezanu, Randy Baden, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Symbiotic relationships in internet routing overlays. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 467–480, Berkeley, CA, USA, 2009. USENIX Association.

[LBSB09] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle inequality variations in the internet. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, IMC '09, pages 177–183, New York, NY, USA, 2009. ACM.

[Lei08] Tom Leighton. Improving performance on the Internet. *ACM Queue: Tomorrow's Computing Today*, 6(6):20–29, October 2008.

[LGS07] Jonathan Ledlie, Paul Gardner, and Margo Seltzer. Network coordinates in the wild. In *Proceedings of USENIX NSDI 07*, Cambridge, Massachusetts, April 2007.

[LMVH07] Jason Liu, Scott Mann, Nathanael Van Vorst, and Keith Hellman. An open and scalable emulation infrastructure for large-scale real-time network simulations. In *Proceedings of the 26th IEEE International Conference on Computer Communications.*, INFOCOMM'07, pages 2476–2480. IEEE, 2007.

[LOH+10] F. Lehrieder, S. Oechsner, T. Hossfeld, Z. Despotovic, W. Kellerer, and M. Michel. Can P2P-Users Benefit from Locality-Awareness? In *Proceedings of the P2P'10: Proceedings of the 10th IEEE International Conference on Peer to Peer Computing*, pages 1 –9, August 2010.

[LR] Skype Limited Lars Rabbe. Cio update: Post-mortem on the skype outage. `http://blogs.skype.com/en/2010/12/cio_update.html`.

[LS08] Cristian Lumezanu and Neil Spring. Measurement manipulation and space selection in network coordinates. In *Proceedings of the 2008 The 28th International Conference on Distributed Computing Systems*, ICDCS '08, pages 361–368, Washington, DC, USA, 2008. IEEE Computer Society.

[MA11] Milton Mueller and Hadi Asghari. Deep Packet Inspection and Bandwidth Management: Battles over BitTorrent in Canada and the United States. In *39th Research Conference on Communication, Information and Internet Policy (TPRC 2011)*, Arlington, Virginia, USA, September 2011.

[MC00] W. Matthews and L. Cottrell. The PingER project: active Internet performance monitoring for the HENP community. *Communications Magazine, IEEE*, 38(5):130 –136, May 2000.

[MCM+11] G. Marchetto, L. Ciminiera, M.P. Manzillo, F. Risso, and L. Torrero. Locating equivalent servants over p2p networks. *Network and Service Management, IEEE Transactions on*, 8(1):65 –78, march 2011.

[Mil67] Stanley Milgram. The small-world problem. *Psychology Today*, 1(1):61–67, 1967.

[MK0] MIT-King-Datensatz, `http://pdos.csail.mit.edu/p2psim/kingdata`, accessed 19. November 2011.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. Springer Berlin / Heidelberg, 2002.

[MSS06] Yun Mao, Lawrence K. Saul, and Jonathan M. Smith. Ides: An internet distance estimation service for large networks. *IEEE Journal on Selected Areas in Communications*, 24(12):2273–2284, 2006.

[MUF+10] Wolfgang Mühlbauer, Steve Uhlig, Anja Feldmann, Olaf Maennel, Bruno Quoitin, and Bingjie Fu. Impact of routing parameters on route diversity and path inflation. *Computer Networks*, 54(14):2506–2518, December 2010.

[MWA02] R. Mahajan, D. Wetherall, and T. Anderson. Understanding bgp misconfiguration. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–16. ACM, 2002.

[Nie98] Jakob Nielsen. Nielsen's Law of Internet Bandwidth, 1998. `http://www.useit.com/alertbox/980405.html`.

[Nit09] Thomas Nitsche. Konsistente Bewertung von Daten in verteilten Systemen am Beispiel eines verteilten Videorecorders, 2009. Diplomarbeit, Technische Universität München.

[NJ12] Kathleen Nichols and Van Jacobson. Controlling queue delay. *Queue*, 10(5):20:20–20:34, May 2012.

[NZ02] T. S. E. Ng and Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proceedings of the Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) 2002*, New York, June 2002.

[OBL+04] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi. Video coding with h.264/avc: tools, performance, and complexity. *Circuits and Systems Magazine, IEEE*, 4(1):7 – 28, October 2004.

[oPU] The Trustees of Princeton University. Planetlab. `http://www.planet-lab.org/`.

[PACR02] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of HotNets–I*, Princeton, New Jersey, October 2002.

[Pax97] Vern Paxson. End-to-end internet packet dynamics. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '97, pages 139–152, New York, NY, USA, 1997. ACM.

[PFA+10] Ingmar Poese, Benjamin Frank, Bernhard Ager, Georgios Smaragdakis, and Anja Feldmann. Improving content delivery using provider-aided distance information. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 22–34, New York, NY, USA, November 2010. ACM.

[PLS05] Peter Pietzuch, Jonathan Ledlie, and Margo Seltzer. Supporting network coordinates on planetlab. In *WORLDS'05: Proceedings of the 2nd conference on Real, Large Distributed Systems*, pages 19–24, Berkeley, CA, USA, 2005. USENIX Association.

[PUK+11] Ingmar Poese, Steve Uhlig, Mohamed Ali Kaafar, Benoit Donnet, and Bamba Gueye. IP geolocation databases: unreliable? *ACM SIGCOMM Computer Communication Review*, 41:53–56, April 2011.

[QS04] Dongyu Qiu and R. Srikant. Modeling and performance analysis of bittorrent-like peer-to-peer networks. *ACM SIGCOMM Computer Communication Review*, 34:367–378, August 2004.

[RCKS05] Sean Rhea, Byung-Gon Chun, John Kubiatowicz, and Scott Shenker. Fixing the embarrassing slowness of opendht on planetlab. In *Proceedings of the Second Workshop on Real, Large Distributed Systems*, WORLDS '05, pages 25–30, Berkeley, CA, USA, 2005. USENIX Association.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, November 2001.

[RD10] Rodrigo Rodrigues and Peter Druschel. Peer-to-peer systems. *Communications of the ACM*, 53(10):72–82, October 2010.

[RGK+05] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 73–84, New York, NY, USA, 2005. ACM.

[Rip01] M. Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *P2P'01: Proceedings of the First International Conference on Peer-to-Peer Computing*, pages 99–100. IEEE Computer Society Press, August 2001.

[RMK+09] Venugopalan Ramasubramanian, Dahlia Malkhi, Fabian Kuhn, Mahesh Balakrishnan, Archit Gupta, and Aditya Akella. On the treeness of internet latency and bandwidth. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09, pages 61–72, New York, NY, USA, 2009. ACM.

[RWHM03] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), March 2003. Obsoleted by RFC 5389.

[Ré08] Réseaux IP Européens Network Coordination Centre. YouTube Hijacking: A RIPE NCC RIS case study. *RIPE-NCC*, February 2008.

[SAA+99] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: informed internet routing and transport. *Micro, IEEE*, 19(1):50 –59, January 1999.

[San10] Sandvine. Fall 2010 Global Internet Phenomena report, June 2010.

[SB09] Moritz Steiner and Ernst W. Biersack. Where is my peer? evaluation of the vivaldi network coordinate system in azureus. In *NETWORKING '09: Proceedings of the 8th International IFIP-TC 6 Networking Conference*, pages 145–156, Berlin, Heidelberg, 2009. Springer-Verlag.

[Sch10] Dieter Schuster. Erstellung eines Application Layer Multicast Moduls im IGOR Ecosystem, 2010. Diplomarbeit, Technische Universität München.

[SKS09] Jan Seedorf, Sebastian Kiesel, and Martin Stiemerling. Traffic localization for p2p-applications: The alto approach. In Henning Schulzrinne, Karl Aberer, and Anwitaman Datta, editors, *Peer-to-Peer Computing*, pages 171–177. IEEE, 2009.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM '01*, pages 149–160, 2001.

[SMW02] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 133–145. ACM, 2002.

[SNS+10] Jan Seedorf, Saverio Niccolini, Martin Stiemerling, Ettore Ferranti, and Rolf Winter. Quantifying operational cost-savings through alto-guidance for p2p live streaming. In *Proceedings of the Third international conference on Incentives, overlays, and economic traffic control*, ETM'10, pages 14–26, Berlin, Heidelberg, 2010. Springer-Verlag.

[SRC84] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2:277–288, November 1984.

[SSW10a] Y. Schwartz, Y. Shavitt, and U. Weinsberg. On the diversity, stability and symmetry of end-to-end internet routes. In *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, pages 1 –6, march 2010.

[SSW10b] Yaron Schwartz, Yuval Shavitt, and Udi Weinsberg. A measurement study of the origins of end-to-end delay variations. In Arvind Krishnamurthy and Bernhard Plattner, editors, *Passive and Active Measurement*, volume 6032 of *Lecture Notes in Computer Science*, pages 21–30. Springer Berlin / Heidelberg, 2010.

[ST04] Yuval Shavitt and Tomer Tankel. Big-bang simulation for embedding network distances in euclidean space. *IEEE/ACM Transactions on Networking*, 12:993–1006, December 2004.

[ST08] Y. Shavitt and T. Tankel. Hyperbolic embedding of internet graph for distance estimation and overlay construction. *IEEE/ACM Transactions on Networking*, 16(1):25 –36, February 2008.

[SW05] R. Steinmetz and K. Wehrle. *Peer-to-peer systems and applications*. Lecture notes in computer science. Springer, 2005.

[Sys06] Systems Research at Harvard (Syrah). King-Blog-Dataset, accessed 12. June 2011. `http://www.eecs.harvard.edu/~syrah/nc`, 2006. Variant with at least 8 neighbors was picked.

[Tec] Akamai Technologies. Akamai sureroute. `http://www.akamai.com/dl/feature_sheets/fs_edgesuite_sureroute.pdf`.

[Tel] Teleography.com. Submarine cable map 2008. `http://www.telegeography.com/`.

[TH09] I.J. Taylor and A.B. Harrison. *From P2P and grids to services on the web: evolving distributed communities*. Computer communications and networks. Springer, 2009.

[TKLB07] Wesley W. Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *Proceedings of the 2007 ACM SIGCOMM Conference*, August 2007.

[VA07] Skype Limited Villu Arak. What happened on august 16, 2007. `http://heartbeat.skype.com/2007/08/what_happened_on_august_16.html`.

[Var01] András Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference*, ESM '01, Prague, Czech Republic, 2001.

[VB00] A. Veres and M. Boda. The chaotic nature of tcp congestion control. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1715 –1723 vol.3, March 2000.

[VLO09] Q.H. Vu, M. Lupu, and B.C. Ooi. *Peer-to-Peer Computing: Principles and Applications*. Springer, 2009.

[Vuz] Inc Vuze. Vuze: The most powerful bittorrent app on earth (website). `http://www.vuze.com/`.

[Wei09] Michael Weiß. Analyse der BitTorrent Cooperation Engine, 2009. Guided Research, Abschlussbericht, Technische Universität München.

[Wil09] Jonathan Will. NAT Traversal Techniques for the IGOR Overlay, 2009. Diplomarbeit, Technische Universität München.

[WMW+06] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet path performance. In *Proceedings of the ACM SIGCOMM '06 Conference*, SIGCOMM '06, pages 375–386, New York, NY, USA, September 2006. ACM.

[WSS05] Bernard Wong, Aleksandrs Slivkins, and Emin Gün Sirer. Meridian: a lightweight network location service without virtual coordinates. In *Proceedings of the ACM SIGCOMM '05 Conference*, SIGCOMM '05, Philadelphia, Pennsylvania, August 2005.

[XYK+08] Haiyong Xie, Yang Richard Yang, Arvind Krishnamurthy, Yanbin Liu, and Avi Silberschatz. P4p: Provider portal for applications. In *Proceedings of ACM SIGCOMM*, Seattle, WA, August 2008.

[ZLMZ05] Beichuan Zhang, Raymond Liu, Daniel Massey, and Lixia Zhang. Collecting the internet as-level topology. *ACM SIGCOMM Computer Communication Review*, 35:53–61, January 2005.

[ZLPG05] Han Zheng, Eng Lua, Marcelo Pias, and Timothy Griffin. Internet routing policies and round-trip-times. In Constantinos Dovrolis, editor, *Passive and Active Network Measurement*, volume 3431 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin / Heidelberg, 2005.

[ZYC09] Changwang Zhang, Jianping Yin, and Zhiping Cai. Rsfb: a resilient stochastic fair blue algorithm against spoofing ddos attacks. In *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, pages 1566 –1567, sept. 2009.

[ZYCC10] Changwang Zhang, Jianping Yin, Zhiping Cai, and Weifeng Chen. Rred: robust red algorithm to counter low-rate denial-of-service attacks. *Communications Letters, IEEE*, 14(5):489 –491, May 2010.

ISBN 3-937201-29-7
ISSN 1868-2634 (print)
ISSN 1868-2642 (electronic)
DOI: 10.2313/NET-2012-05-1