ERROR RESILIENT AND ROBUST OVERLAY NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Dragan Milić

von Serbien

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

ERROR RESILIENT AND ROBUST OVERLAY NETWORKS

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Dragan Milić

von Serbien

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik und angewandte Mathematik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 26.05.2010

Der Dekan:
Prof. Dr. Urs Feller

Contents

List of Figures

Preface

This thesis is a result of the the work performed during my employment as research assistant at the Institute of Computer Science and Applied Mathematics (IAM) of the University of Bern. The research presented here was mainly conducted within the Swiss National Science Foundation project ERON and European Union project EU-QoS.

I would like to thank everybody who provided me with support, ideas, and encouragement during my employment at the Computer Networks and Distributed Systems group (RVS). My special thanks go to Dr. Torsten Braun for supervising the work and for providing guidance, advice and patience that made this work possible. I would also like to thank Prof. Dr. Ulrich Ultes-Nitsche for having accepted to read and judge this work, and Prof. Dr. Gerhard Jäger who was willing to be the co-exterminator of this work.

Furthermore I would like to express my thanks to my colleagues at the IAM for very helpful hints, enlightening discussions and for being a part of great team I will be missing. Special thanks go to Matthias Scheidegger, Marc Brogle, Florian Baumgartner, Peppo Brambilla, Markus Anwander, Markus Wälchli, Thomas Staub, Thomas Bernoulli, Gerald Wagenknecht and Markus Wulff. I would also like to thank our our secretary Ruth Bestgen for her excellent work and great support.

Many thanks go to my friends and relatives Slavica Milić, Zlatomir Milić, Teodora Tičerić, Igor Djordjević, Nenad Perić, Bogdan Kecman and Corinne Husi for their support and for being there for me. I would also like to thank Elke Hentschel for her support and encouragement to pursue an academic career.

Last but not least, I would like to express my thanks to Cindy-Jane Armbruster for proofreading this thesis, correcting both language and stylistic errors and pointing out inconsistencies in the original manuscript. I would also like to thank her for her support, understanding and sharing a wonderful time with me.

Chapter 1

Introduction

Overlay or peer-to-peer (P2P) networks can be very useful when introducing new or missing services into an existing network. This is the case where a new service is still in development, or deploying the new service within the underlying network is not possible. To communicate between end systems, every overlay network uses an already existing communication network, e.g. the Internet. We denote this existing communication network as the underlying network. To introduce new services, overlay networks rely exclusively on basic functionality provided by the underlying network: unicast (one-to-one) exchange of messages between end systems.

Traditionally, overlay networks are used to provide services such as distributed hash table (DHT), overlay multicast, overlay anycast, file sharing or finding QoS paths within the network.

No matter what service is provided by the overlay network, it is important to be aware of the topology of the underlying network, both when the overlay network is built, and also when so-called routing decisions are made, i.e., decisions about which neighbors to use to route a message. If the topology of the underlying network is not taken into account when the overlay network is created, then a "short" route in the overlay network can result in a rather long route in the underlying network. Such topology-unaware overlay networks lead to an ineffective use of network resources as well as larger than necessary delays of service operations resulting in bad service performance.

Unfortunately, most of the existing overlay networks totally disregard the topology of the underlying network. The main aim of this dissertation is to develop the protocols and mechanisms necessary to construct robust and topology-aware overlay networks. To achieve this goal, we explore round-trip time (RTT) prediction schemes and their uses to construct topology-aware overlay networks. We use those prediction schemes as a basis for different services such as unicast routing, multicast routing, service discovery, etc.

Before giving an outline of the main contributions of this thesis, in this Section, we clarify the terminology we will be using in this dissertation. Next, we give a short introduction to RTTs and why we prefer using RTTs rather than communication delays. Then, we introduce a graph abstraction of the underlying and the overlay network from the perspective of the end systems.

1.1 Terminology

1.1.1 Computer Networks

In general, a packet-switched network such as the Internet consists of end systems and routers. Figure 1.1 shows all components of a packet-switched network.

- **Router:** A router is a network device that is connected to other routers and end systems. Each router has several network interfaces and is able to forward incoming network packets from one network interface to either one other network interface (in the case of unicast transmissions) or to several other interfaces (multicast). Routers are usually owned by Internet Service Providers (ISPs). As a result, changing the software on routers, e.g. in order to introduce new services, is a lengthy process and requires the cooperation of all service providers in the Internet. This is one of the reasons, why, to date, IP Multicast is not widely available in the Internet.

- **End System:** An end system is usually a computer with a connection to one of the routers. Unlike routers, end systems are normally not involved in forwarding network packets. Each end system is usually controlled by the person operating it. Consequently, on an end system, it is the individual user who deploys new software. Another difference is that end systems tend to have more resources at their disposal, such as computer memory and available CPU speed; however, this is not always the case, e.g. embedded devices or mobile devices.

- **Physical Network Link:** Physical network links represent layers 1 and 2 of the OSI reference network model. This can be any type of Ethernet link, ATM, ISDN, serial line, ADSL, VDSL, or any other physical link which connects two or more network components and is used to transfer data.

- **Logical Network Link:** For our purposes, a logical network link is a path between two end systems through the underlying network. A logical network link consists of at least one, but usually several, physical network links which are interconnected by routers. Since end systems are usually not aware of the topology of the underlying network, logical network links provide an abstraction of how end systems are interconnected. A logical network link is characterized by the collective characteristics of the physical network links underlying it. For example, the maximal available bandwidth of a logical network link is the smallest bandwidth available on its component physical network links. Another example is a logical link's minimal RTT, which corresponds to the sum of all RTTs of the underlying physical network links (plus the processing time of each router along the path).

1.1.2 Graph Theory

In general, any computer network can be represented as a graph. In this dissertation, we will be referring to both, underlying and overlay networks, as graphs because this enables

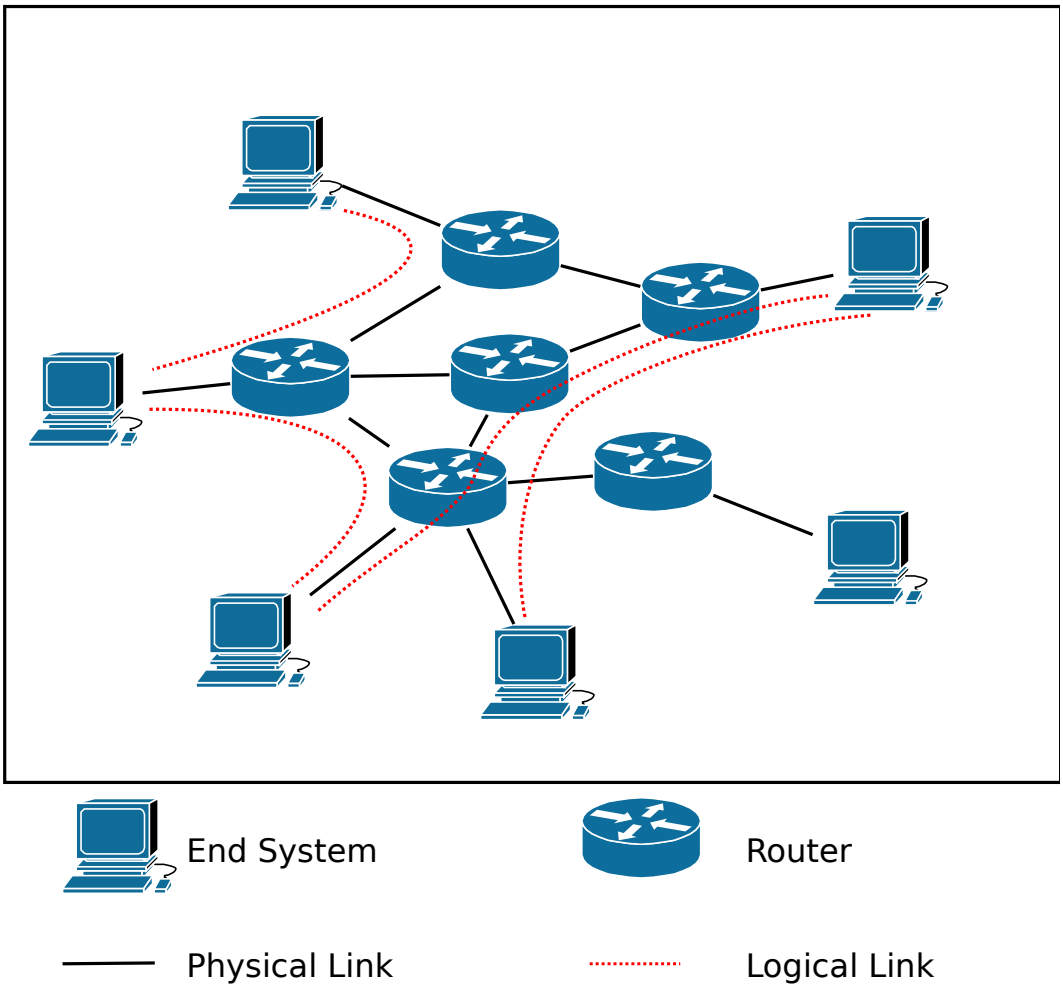

Figure 1.1: Example of a Computer Network (Internet) and Its Components (End Systems, Routers, Physical and Logical Links)

us to define various network problems as graph theoretical problems, e.g. a routing problem becomes a problem of finding a path in a graph). The mathematical definition of a graph is as follows: A graph $G := \{V, E\}$ consists of a set of vertices V and a set of edges E. Each edge connects two vertices. We denote an edge between the two vertices v and w as $v - w$. If edges are "weighted", i.e., each edge has a number assigned to it, the resulting type of graph is a weighted graph. Additionally, graphs can be "directed". Here, the edge $v \rightarrow w$ leads only in one direction from vertice v to vertice w. If the same graph also contains the connection from vertice *w* to vertice *v*, then it requires another edge ($w \rightarrow v$) representing that direction. If none of the edges of a graph are directed, the graph is undirected, otherwise it is directed.

In this thesis, we use the following concepts and terminology from graph theory:

- **Sub Graph:** A sub graph G' to a graph G is constructed by removing at least one of the edges of G.
- **Clique:** A clique (denoted by $G*$) is a graph in which all vertices are directly connected by an edge. A clique is also called a maximal graph, as for the given vertices, all possible edges are present. Every graph for a given set of vertices *V* is a sub graph of the clique of *V*.
- **Path:** A path is a sequence of vertices $v_1, v_2, v_3, \ldots, v_n$ in *G* such that there is always an edge $v_{i-1} - v_i$ in *E*. A path is cycle-free if and only if every vertex occurs at most once in the path.
- **Path Cost:** For a cycle-free path, the path cost is the sum of the weights of the edges created by each pair of vertices in the path's sequence.
- **Cycle-Free Graph:** A cycle-free graph is a graph in which every path is cycle-free.
- **Connected Graph:** A graph *G* is a connected graph if and only if there is a path for every pair of vertices *v* and *w*.
- **Tree:** A tree is a connected cycle-free graph.
- **Vertex Degree:** We denote the number of edges connecting to a vertex as its degree. In directed graphs, each vertex has an inbound degree, i.e., the number of edges connecting to it as well as an outbound degree, i.e., the number of edges connecting that vertex to other vertices.
- **Dense and Sparse Graphs:** Whether a graph is sparse or dense depends on how many edges it contains. A graph is dense if the number of its edges is proportional to its maximal possible number of edges (i.e., the number of edges of the clique). In an undirected graph with $|V|$ vertices, the maximally possible amount of edges is $\frac{|V| \cdot (|V|-1)}{2}$. In a directed graph, the highest possible amount of edges is $|V|^2 - |V|$.

1.2 Graph Abstraction of a Computer Network

The most natural way to depict the computer network from Figure 1.1 as a graph is to represent each router and end system as a vertex. Since this dissertation is mainly interested in communication delays, we indicate the RTT of every physical link as the weight of each edge. This results in a weighted directed graph. One example of such a graph is shown in Figure 1.2.

There are two problems with such a naive approach to graph representation of a network. Firstly, there is the problem of measuring the one-way delays within the network. As discussed in the previous Section, this problem is nontrivial and cannot be solved in a practical way which would scale in a large and distributed environment such as the Internet. The second problem is that this representation of a network requires intimate knowledge of

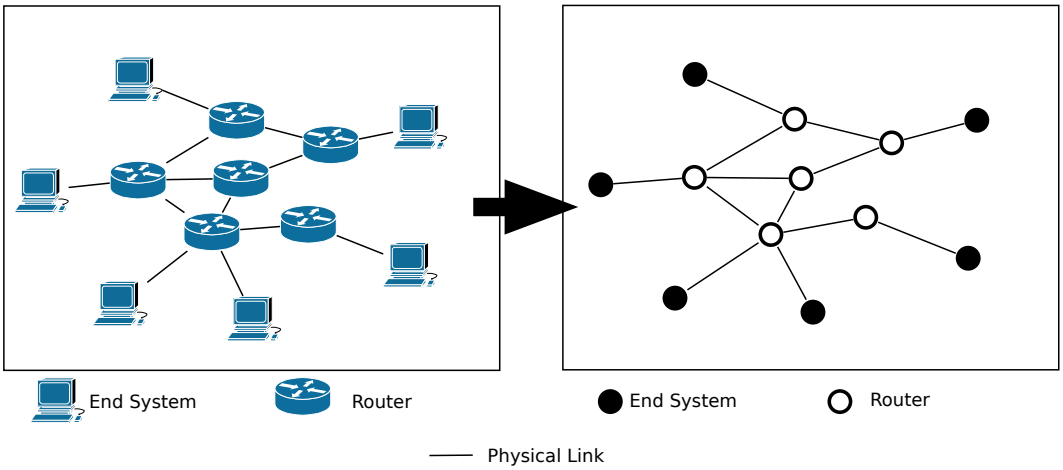

Figure 1.2: Mapping a Computer Network to a Graph by Using Physical Links as Edges

every router and every data link in the Internet. Almost every ISP considers this information to be part of its trade secret and is not willing to share it with other parties. We could attempt to overcome this problem by using tools such as trace route [1] to infer the topology of the underlying network. However, we would still be facing the first problem of not being able to measure the one-way delay of every physical link. In addition to those problems, there are the dynamics of the underlying network: Physical links and routers occasionally fail. Routing protocols take this into account and react to such outages by rerouting messages along other paths. This means that even if we had a graph precisely representing a specific network, e.g. the Internet, circumstances would change too quickly to keep up with them and we would soon end up with a graph that bears hardly any resemblance with the current state of the network.

Even though it is virtually impossible to construct a precise graph of the Internet, we can still have an abstraction which represents the point of view of an end system. End systems are usually unaware of the rest of the network. All they do is forwarding messages to the router they are directly connected to and trust that these messages are most probably delivered at their destination. They do not know how many routers and data links are involved in routing messages. Each end system only knows that if the end system it communicates with is running and connected to the network, then its messages will (most probably) be delivered. So, from the point of view of an end system, the Internet (or any other packet-switched network) is a clique (i.e., a graph where every pair of vertices is “virtually“ directly connected by an edge) consisting of all the end systems which are connected to it. In this dissertation, we assume that the connectivity of end systems is not restricted by anything, e.g. firewalls or network address translators (NATs). This means that every end system is able to communicate directly with any other end system, provided that the network address of that end system is known.

While we have just solved the problem of the unknown underlying structure of a net-

work, we are still faced with the problem of not being able to determine the communication delay between end systems if they do not have synchronized clocks. However, we can relatively easily determine the round-trip time. The RTT is the sum of the delays incurred en route along the logical network link from end system a to end system b and back from end system b to end system a. Please note that the delay on the logical link from a to b is not necessarily the same as the delay on the logical link from b to a. This is due to the fact that not every data link in the Internet is bidirectional, meaning that the optimal path from a to b may include completely different nodes than the optimal path from b to a. Additionally, in the Internet, the most commonly used routing protocol BGP [2] allows for "policies". Policies let ISPs redirect Internet traffic according to financial agreements they have with peering ISPs. At the same time, the use of policies results in BGP not always choosing optimal routes through the Internet. Despite all this, RTTs give us a notion of the "distance" between two end systems, which we can use as a weight in a graph.

Combining these two ideas, i.e., considering the network to be a clique as seen from the point of view of an end system and using RTTs instead of one-way delays, we come up with the following representation: From the point of view of an end system, the Internet can be represented as a weighted clique, i.e., a full graph of end systems using RTTs as the weights of the edges. We show this approach in Figure 1.3. This way of representing

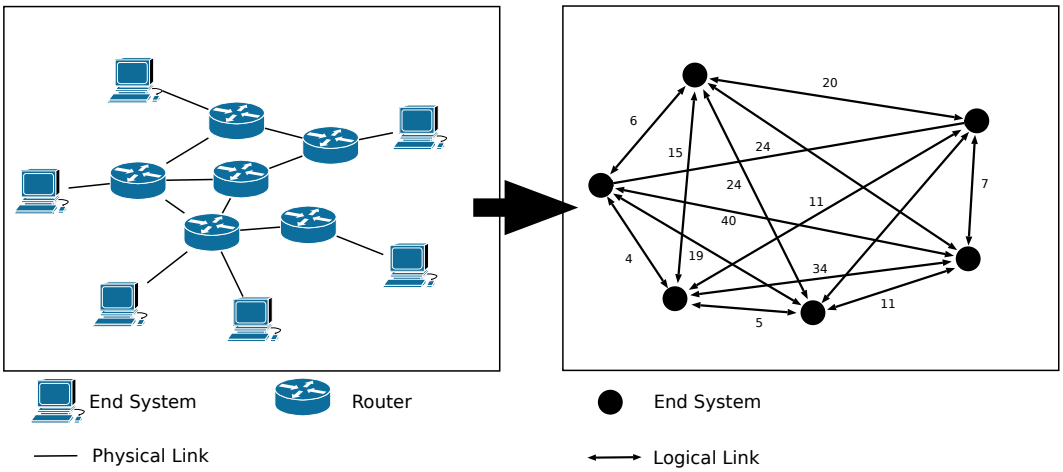

Figure 1.3: Mapping a Computer Network to a Clique by Using Logical Links as Edges and RTTs as Weights

the underlying network is also known as the black box view. Here, end systems do not try to infer the topology of the underlying network. Instead, they only measure the properties they are interested in, e.g. RTTs, jitter, bandwidth, etc.).

There are several different ways of representing a graph. A very handy one is the representation as a matrix. To do so, we assign an index to all vertices of the graph, starting with 1. Like that we enumerate all vertices from $1 \ldots |V|$. Thus, the graph is represented as a $|V| \times |V|$ matrix, where for each pair of vertices v and w this matrix contains the weight

$d_{v,w}$. One example of such a matrix is:

$$\begin{pmatrix} 0 & d_{1,2} & d_{1,3} & \cdots & d_{1,|V|-2} & d_{1,|V|-1} & d_{1,|V|} \\ d_{2,1} & 0 & d_{2,3} & \cdots & d_{2,|V|-2} & d_{2,|V|-1} & d_{2,|V|} \\ d_{3,1} & d_{3,2} & 0 & \cdots & d_{3,|V|-2} & d_{3,|V|-1} & d_{3,|V|} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ d_{|V|-2,1} & d_{|V|-2,2} & d_{|V|-2,3} & \cdots & 0 & d_{|V|-2,|V|-1} & d_{|V|-2,|V|} \\ d_{|V|-1,1} & d_{|V|-1,2} & d_{|V|-1,3} & \cdots & d_{|V|-1,|V|-2} & 0 & d_{|V|-1,|V|} \\ d_{|V|,1} & d_{|V|,2} & d_{|V|,3} & \cdots & d_{|V|,|V|-2} & d_{|V|,|V|-1} & 0 \end{pmatrix}$$

Since RTTs are symmetrical, i.e., the RTT from a to b is the same as the RTT from b to a, the values of the weights $d_{v,w}$ and $d_{w,v}$ are same, resulting in a symmetrical matrix.

1.3 Overlay Networks

As we have shown in Section 1.2, from the point of view of an end system, the Internet can be represented as a weighted clique (complete graph) of end systems, where the RTTs between end systems are the weights. Storing such a graph on each end system is not feasible, since the amount of information to be stored grows quadratically with the number of end systems involved. In order to have a scalable overlay network, where the number of end systems is not limited by the memory available on each individual end system, each end system stores information about only a part of the overlay network. This implies that an overlay network is a sub graph of the clique representing all end systems of the underlying network. In the general case, an overlay network is a directed weighted graph that is a sub graph of the underlying network as we depict it in Figure 1.4.

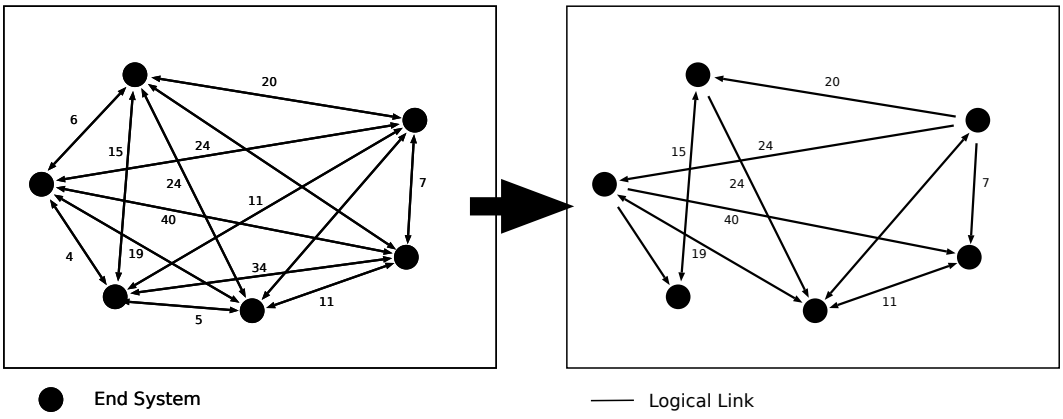

Figure 1.4: An Overlay Network Represented as a Directed Weighted Sub Graph

Overlay networks are constructed by end systems selecting their neighbors. Neighbors of an end system (vertex) are those end systems which are connected to the original end

system by outgoing edges in the overlay network graph. In other words, all those end systems directly known to one end system constitute its neighbors.

Which end systems are chosen as neighbors has a direct impact on the performance of the overlay network. This choice influences which paths are available in the overlay network to connect any two end systems. This choice also decides if the graph of the overlay network is connected (cf. 'connected graph' in Section 1.1.2). If the overlay network is not connected, then there will be end systems which cannot communicate with each other through the overlay network. This would lead to the exclusion of some end systems from services provided by the overlay network or could even split the overlay network into two completely separate overlay networks.

In an overlay network, not every pair of end systems is directly connected to each other. Therefore, a message sent from end system a to end system b has to make its way through the overlay network, sometimes involving intermediary end systems. Such intermediary systems are also called relays. The choice of the next relay for a given message is called a routing decision. Depending on which relay is chosen, the path through the graph representing the overlay network is more or less optimal in terms of the amount of hops taken and the total RTT of the path. To make sure the optimal route is taken, we could use some of the existing routing protocols such as [3–5]. The problem of this approach is that, in order to make the routing decision, each end system would need too know the complete graph of the overlay network. This defeats the purpose of trying to minimize the size of the routing table stored on each end system.

Most overlay routing protocols such as [6–8] completely disregard the weights of the graph's edges (i.e., the RTTs of the links). They simply try to minimize the number of hops taken to reach the destination. On average, this leads to disappointingly large message delivery delays. Also, by unwittingly using "long" links, this routing strategy imposes additional load on the underlying network since the same message may end up traveling along many physical network links, some of them even more than once.

A topology-aware overlay network should take into account at least RTTs as an optimization parameter. Minimizing the RTT on a path between end systems a and b, minimizes the use of network resources for data transmission. As we will show in chapter 7, no optimization is ever as effective as finding an optimal path in a graph representing the complete overlay network. However, a fairly good balance can be achieved between the scalability of the approach, the overhead to perform the optimization, and the obtained results.

Of course, other QoS parameters of a network, such as the available bandwidth, jitter, etc., may also be considered when planning an overlay network. The problem with those parameters is that they are not additive. For example, if we denote the bandwidth between a and b as B_{a-b}, the bandwidth between b and c as B_{b-c} and the bandwidth between a and c as B_{a-c}, the bandwidth on the path (a, b, c) is $\min(B_{a-b}, B_{b-c})$. This makes bandwidth a very unsuitable candidate for a distance function. On the other hand, high RTTs usually correlate with low available bandwidth. Conversely, high bandwidth usually does not imply low RTTs. This means that optimizing for RTT (i.e., minimizing RTT) usually also means optimizing for available bandwidth on the path.

1.4 Round-Trip Time vs. One-Way Delay

The first question to ask when confronted with analyzing properties of round-trip times (henceforth referred to as RTT) in a packet-switched network is the following: Why measure RTTs and not the one-way delay? The advantage of RTTs is that they are easily measured, whereas the exact one-way delay between two network components can only be obtained if the clocks are synchronized. To measure a delay between two network components a and b, we need to send at least one message M from a to b. This message has to contain the exact point in time T_1 at which the message was sent. Upon receipt of the message, b has to compare its local time T_2 to the time T_1 it received with the message. The difference $T_2 - T_1$ is the communication or one-way delay. This approach fails as soon as a and b do not use the same time base, i.e., as soon as the clock in a is not exactly synchronized to the clock in b. If the clocks in a and b show different times, the calculated message delay will be systematically wrong. Even worse is the case in which the clocks in a and b do not advance at the same speed (i.e., time skew); here, each measurement of the communication delay will obtain a different result.

There are different ways to solve this problem and to obtain synchronized clocks. However, none of them is particularly practicable. For example, each node could use a high precision atomic clock. Alternatively, GPS [9] could be used to synchronize the clocks as it offers a relatively precise time service. Both of these options are impractical due to the high costs involved per node as well as other issues, such as requiring good GPS signal reception at each node in order to provide a network service. The third and most commonly used option is the network time protocol [10]. Using NTP is more practical, however, its precision is too low to reliably measure the delays in most modern data links.

Unlike measuring one-way delays, measuring the RTT between a and b is relatively easy: In order to measure the RTT between the two network components a and b, a sends a message $M^a_{T_a}$ to b. The exact point in time T_1 at which the message was sent is either contained within the message or stored in a. Upon receipt of message $M^a_{T_1}$, b responds immediately by sending a message $M^b_{T_1}$. When a receives message $M^b_{T_1}$, it compares the current time (T_2) to the time point when the original message was sent (T_1) and is thus able to calculate the time it took the message to make its round trip (RTT): $T_2 - T_1$. Since only the clock of a was used in this operation, there is no need for synchronized clocks in all network components. The downside of this approach is that the RTT gives no indication of how much time was needed for the individual legs of the trip – i.e., how long it took to get from a to b, which may differ significantly to the delay from b to a.

1.5 Round-Trip Time Predictions

Next to the available bandwidth, the round-trip time is one of the most important measurable properties of Internet communication. This is particularly due to its impact on the effectively available bandwidth for TCP connections [11]. Information on RTTs can be used to help choosing an FTP or an HTTP mirror, to optimize overlay and peer-to-peer

structures, to maintain quality of service (QoS) routing, etc.

To obtain information on RTTs, we could use a naive approach where we measure the RTT between every pair of end systems in the Internet. As shown above, the overhead for both measuring and storing such information is quadratic ($O(|V|^2)$). In other words, for small $|V|$, it is possible to perform these measurements; for bigger $|V|$, however, the required storage space increases very rapidly.

1.5.1 Round-Trip Time as a Distance

A distance function numerically quantifies what is colloquially known as the distance between two objects. In mathematics, a distance function is a function in a metric space which quantifies the distance between any pair of elements in that space. More formally, a metric space is a 2-tuple $(A, d_{\mathcal{S}})$ where A is a set and $d_{\mathcal{S}} : A \times A \rightarrow \mathbf{R}$ is a distance function. A function $d_{\mathcal{S}}$ is called a distance function if the following conditions hold for all $a, b, c \in A$:

$$\begin{aligned} d_{\mathcal{S}}(a,b) &\geq 0 \\ d_{\mathcal{S}}(a,b) = 0 &\Leftrightarrow a = b \\ d_{\mathcal{S}}(a,b) &= d_{\mathcal{S}}(b,a) \\ d_{\mathcal{S}}(a,c) &\leq d_{\mathcal{S}}(a,b) + d_{\mathcal{S}}(b,c) \end{aligned} \tag{1.1}$$

A k-dimensional Euclidean space is a special case of a metric space with

$$A := \mathbf{R}^k$$

$$d_{\mathcal{S}}(a,b) := \sqrt{\sum_{i=1}^{d} (a_i - b_i)^2}$$

The inequality (1.1) is also known as the triangle inequality. In other words, there exists no triplet of elements $a, b, c \in A$ where the distance between a and c is greater than the sum of the distance between a and b and the distance between b and c. In a Euclidean space, one consequence of the triangle inequality is that we can always construct a triangle using the distances between three points. This includes the pathological case where the triangle is a one-line segment ($a + b = c$). If three distances do not fulfill the triangle inequality, for example: 1, 3 and 5, they cannot be arranged into triangle, since there are no three points in any metric space where the distances would match.

It is important to realize that in almost all situations, we do not need to know the exact RTT. Approximate values for each pair of end systems suffice entirely. One way of obtaining such values is to treat RTTs as an indication of distance in the network. Indeed, RTTs share many properties with distance functions in a metric space. They are never negative. They are symmetric, meaning that the RTT between end systems a and b is the same as between b and a. Also, the RTT from a to a is 0 or at least can be considered to be 0. The only condition of a distance function which RTTs do not always fulfill is the triangle inequality.

1.5.2 Triangle Inequality Violations

There are two main reasons why RTTs do not always fulfill the triangle inequality. First, in the Internet, routes are optimized for a minimal number of hops taken (number of routers involved) rather than for minimal delay along the paths. For short physical network links (short physical distance), this is usually not a problem; however, in the case of long physical network links, this becomes a problem, because choosing one link over another can have substantial consequence on the data delivery delay.

The second reason why RTTs in the Internet do not fulfill the triangle inequality lies in the nature of BGP routing. BGP routing allows each ISP to have different policies, each of which influences the path taken by a packet from a to b. In many cases, these policies result in less than optimal paths. Those can be avoided by using a separate end system as a relay [12, 13]. In other words, taking a "detour" over a third end system may result in a shorter path (less delay) than using direct routing, which is a clear violation of the triangle inequality.

1.5.3 Modifying Measured Round-Trip Times to Fit Into a Metric Space

If RTTs were metric we could represent them as distances in a metric space. We would then be able to assign positions to end systems within that metric space so that the distances between those positions represent RTTs. The distance matrix could then be replaced by a matrix storing only the positions of the end systems. This would result in a significantly reduced need for storage space. Unfortunately, because RTTs do not always fulfill the conditions of a distance function in a metric space, in general, we will not be able to find a metric space where RTTs can be represented as a distance. However, if we are allowed to "adjust" the RTT information, we may "make" it metric.

The most commonly used metric space is a k-dimensional Euclidean space. In a k-dimensional Euclidean space, every point $\mathcal{C}$ is defined as a k-tuple: $(c_1, \ldots, c_k)$ with c_m being real numbers. The distance function in a Euclidean space is defined as:

$$d(\mathcal{C}_i, \mathcal{C}_j) = \sqrt{\sum_{m=1}^{k} (c_m^i - c_m^j)^2}. \tag{1.2}$$

If we know the positions of the end systems, we can calculate all distances. Hence, a matrix listing the positions of all end systems is an alternative representation of the distance matrix. If we can keep k significantly lower than the number of end systems $|V|$, our alternative representation of distances requires much less space. Keeping k constant, the complexity for storing distances in such a way are linear ($O(n)$). Compare this to the quadratic ($O(n^2)$)

required by the distance representation.

$$\begin{pmatrix} 0 & d_{1,2} & \cdots & d_{1,|V|-1} & d_{1,|V|} \\ d_{2,1} & 0 & \cdots & d_{2,|V|-1} & d_{2,|V|} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{|V|-1,1} & d_{|V|-1,2} & \cdots & 0 & d_{|V|-1,|V|} \\ d_{|V|,1} & d_{|V|,2} & \cdots & d_{|V|,|V|-1} & 0 \end{pmatrix} \Longleftrightarrow \begin{pmatrix} c_1^1 & c_2^1 \\ c_1^2 & c_2^2 \\ \vdots & \vdots \\ c_1^{|V|-1} & c_2^{|V|-1} \\ c_1^{|V|} & c_2^{|V|} \end{pmatrix}$$

As mentioned above, the problem of this approach is that RTTs are not guaranteed to be metric. In order to make them metric, we have to adjust the original RTT matrix. These changes must be as small as possible so that we introduce only as little error as is needed to obtain a Euclidean distance matrix. This process of adjusting RTTs so that they are distances in an metric space is also known as embedding. Quantifying the introduced error is not a easy task, since we are "allowed" to arbitrarily change any of the distances in the matrix, as long as we do not violate other distance conditions (i.e., distance must be positive and zero only from a point to itself; it must also be symmetric and fulfill the triangle inequality). In order to evaluate how much error was introduced by our changes, usually the sum of squared error function (1.3) is used:

$$\mathcal{E} = \sum_{i,j \in \{1,\ldots,|V|\}} (d(\mathcal{C}^i, \mathcal{C}^j) - d_{i,j})^2 \tag{1.3}$$

The smaller this sum, the smaller is the error introduced by the whole embedding. Previous research [14–17] has shown that the error introduced when embedding RTTs measured in the Internet is quite small.

1.5.4 Predicting Round-Trip Times

Under the assumption that RTTs can be represented as distances in a metric space, this also indicates that to position an end system in such a virtual space, we do not need to measure its RTTs to all other end systems in the Internet. Theoretically, all the end system has to do to determine its position in the Euclidean space is measure its "distance" (RTT) to only $k + 1$ other end systems with known positions. This procedure of determining an unknown position-based only on distance measurements is known as multilateration. By measuring the RTTs (distances) to only a few end systems in the Internet we trade in the precision of RTT estimation for the number of measurements needed. In other words, the more measurements to different end systems we perform, the more accurate are the predicted RTTs. On the upside, we have just obtained a way of predicting RTTs, which gives us an idea of the distances between end systems, without having had to perform all the measurements. The RTTs thus obtained are called RTT predictions.

1.5.5 Applications of Embedding Round Trip Times in a Metric Space

Basically, there are two approaches for extracting topology information about the underlying network. The most straightforward method is to consider the underlying network as a

white box. This means that the topology information (topology graph) of the underlying network is extracted and used to construct an optimal overlay network. Topology information gathering is usually done either by querying the databases provided by the ISPs and studying BGP routing tables, or by using topology discovery tools [18, 19]. This approach may yield good results for small overlay networks, because it allows a fairly precise optimization on the physical link level. On the other hand, this approach introduces too much overhead and complexity for large overlay networks. The amount of information needed to optimize the overlay structure of a large overlay network is enormous. In addition, finding the optimal solution for the problem is more or less equivalent to the traveling salesman problem, which is NP-hard. Another issue is that ISPs are rarely willing to provide information about the internal structure of their networks.

In contrast to the white box approach, in the black box approach, the structure of the underlying network is not analyzed. Instead, targeted QoS parameters such as available bandwidth, RTTs, jitter, etc. are actively or passively measured between several end systems. Based on this information, the structure of the overlay network can be optimized. Still, measuring QoS parameters for each pair of end systems scales quadratically $O(n^2)$ relative to the number of end systems. Also, the optimization task is still NP-hard. This makes the optimization of large overlay networks unscalable. To solve this problem, we should not aim for finding the optimal solution. Instead, we need to find a method that provides a solution close to the optimal one, but with much lower cost in terms of number of measurements required and computational overhead.

Embedding end systems into a virtual metric space, e.g. a Euclidean space, opens new avenues for making different services aware of the topology of the underlying network. We now have a very scalable way of predicting RTTs between end systems as well as a virtual metric space in which each end system is represented by one point (position). Such a space also has a concept of direction, a characteristic which is absent in plain graphs. This gives us the opportunity to use location-based algorithms, for example for routing. In chapter 7, we will show how we can use the embedding of end systems to provide topology-aware unicast, multicast or discovery services. We will also show how such embeddings can be used to find paths in the underlying networks which fulfill the requirements of QoS.

1.6 Contributions of this Thesis

In this thesis, we present various contributions to the field of RTT prediction. In a first part, we describe our contributions to improving GNP – an already existing approach for RTT prediction. In addition, we introduce fisheye view, a new method for building topology-aware overlay networks. We also introduce a further development of VIVALDI, called NetICE9, a dynamic and fully distributed RTT embedding system. Compared to VIVALDI, NetICE9 results in much better RTT embedding and a more stable system.

1.6.1 GNP Improvements

Global Network Positioning (GNP) was a pioneering approach to RTT prediction. It proposed that RTTs be considered as a metric and embedded into metric spaces. GNP knows two types of end systems. End systems of the first type are called landmarks. Landmarks are a fairly small number of end systems which are used by all other end systems to position themselves with. Landmark end systems measure the RTTs among each other in order to obtain a complete RTT distance matrix. As soon as this RTT distance matrix is complete, one of the landmark end systems computes the positions of all landmarks and spreads that information to all other landmarks. From now on, all other end systems just measure the RTTs to each of the landmarks and calculate their own position within the virtual space by using the landmarks' positions and the measured RTTs.

The problem of GNP – and of any other approaches embedding end systems in a k-dimensional Euclidean space – is the choice of the number of dimensions (k). If k is too low, the error of the embedding will be too large. The bigger the number of dimensions, the more the error of the embedding decreases; however, at the same time, the computational overhead of the embedding grows.

Both, landmarks as well as regular end systems, are positioned by minimizing the sum of the squared differences between the measured and the predicted RTTs (error minimization). Unfortunately, there is no known closed form for finding the minimum of this function. Hence, the function must be minimized using a numerical algorithm (e.g. [20]), which evaluates the function for different combinations of parameters and tries to find a better solution at each step. In the case of L landmarks and a k-dimensional virtual space, this function has $L \cdot k$ parameters. Our experiments have shown that the computational overhead of such a function minimization increases, on average, cubically ($O(n^3)$), if n is the number of parameters.

The process of finding the optimal number of dimensions for embedding distances into a k-dimensional Euclidean space is very resource intensive. The reason for this is that in order to find the optimal number of dimensions, the embedding for every number of dimensions (from 1 to $L-1$) has to be performed and compared. Simply using the maximal number of dimensions ($k = L-1$) results in a waste of computing resources and in a long computational time to obtain the landmarks' positions. In chapter 3, we present an heuristic method we invented which yields very good results and is not as computationally intensive as the attempt to find the optimal number of dimensions. In the same chapter, we also describe an algorithm designed to find a "good" starting point for the function minimization. Because this starting point is "near" the optimal solution, function minimization is accelerated and requires fewer steps to reach the optimal solution than would be the case with a random starting point.

A second problem of GNP is that the function which is minimized to determine the position of a regular end system usually contains more than one local minimum. If we blindly attempt to minimize such a function, chances are high that we will find one of the local minima instead of the global minimum (correct solution). In chapter 4, we present an algorithm that searches for all local minima of the given function in order to find the global

minimum.

1.6.2 Topology-Aware Overlay Network

Finding the optimal choice of neighbors for each end system is not easy. Ensuring that the graph representing the overlay network is connected and that the vertex degree (described in 1.1.2) is bound by a constant (thus defining the maximal memory requirements on each end system) are no simple tasks.

In our research, we have developed fisheye overlay, an overlay network building strategy, which builds overlay networks in such a way that each end system has a maximum of c neighbors (c is a constant) no matter how many end systems make up the overlay network. Guaranteeing that the resulting overlay network is connected, fisheye overlay constructs an undirected connected graph, i.e., in each end system, the inbound degree is the same as the outbound degree (lower than c). Furthermore, fisheye overlay chooses as neighbors those end systems which are spread geographically diverse around the selecting end system. As the name suggests, the density of the neighbors chosen by fisheye overlay follows the level of detail of the lens of a fish eye. All of these features make any network built by fisheye overlay an ideal overlay network for many purposes; e.g., the network substrate of a topology-aware application-layer multicast service.

Another interesting feature of fisheye overlay is its adaptability to changing network conditions. fisheye overlay periodically re-evaluates the end system's view of its neighbors and constantly tries to build a better view by improving the selection of neighbors. Chapter 5 offers a more detailed discussion of the fisheye overlay algorithm and its communication protocol.

1.6.3 Stable and Precise Round-Trip Time Embedding

Also interesting is the application of a fisheye overlay network as the overlay network used to embed end systems into a metric space. During our research, we noticed that VIVALDI's well-known stability issues when embedding RTTs into a metric space were based in the fact that in VIVALDI, end systems choose their neighbors at random. This choice of neighbors is not symmetrical (in the terms of graph theory, the "is a neighbor of" graph is unidirectional). As a result, in the cases where the triangle inequality was violated, one-way loops were formed in which end systems would "chase" each other through the virtual space. This behavior leads to end systems constantly changing their position in the virtual space even if the underlying network never actually changes.

We discovered that this misbehavior of VIVALDI is reduced if the "is a neighbor of" graph is undirected. We then experimented with VIVALDI using fisheye overlay to select the neighbors. This resulted in far more stable positions of end systems.

Encouraged by these results, we went a step further. We improved how end systems optimize their positions relative to their neighbors by doing every possible optimization at the same time instead of doing them one at a time (as is the case in VIVALDI). Thus, we were able to stabilize the system even further. This approach also dramatically improved the

precision of RTT embedding. It was better than the precision achieved by GNP! We named our approach NetICE9, because it resembles the process of building a crystal structure with end systems as atoms. We describe NetICE9 in chapter 6 of this thesis.

1.6.4 Use of Embedding

In this dissertation, we also present some use scenarios for the fisheye overlay network. We identify why simple, position-based greedy routing is not successful in our fisheye overlay network. We define an overlay network in which greedy routing is always successful. We then combine this overlay network with the fisheye overlay network to obtain a very simple unicast overlay network routing protocol that is simple, scalable and topology-aware.

We also define a topology-aware overlay multicast protocol. This application-layer protocol uses the fisheye overlay as its substrate and a flooding mechanism to find a distribution tree for every sender. Each such distribution tree is a sub graph of the fisheye overlay network.

Based on our scalable unicast greedy routing protocol, we also define a service discovery protocol which is able to flood the overlay network in order to find those end systems that offer a specific service, such as specific data or sufficient CPU speed or memory resources to perform a computation. Our service discovery protocol also allows for spatially limited queries. This means that each query can be limited to a specified portion of the virtual space.

Limiting queries to a certain part of the virtual space allowed us to define a service for finding QoS paths through the overlay network for any pair of end systems. Essentially, this service performs a recursive search, starting from one end system and involving all end systems which are within the RTT boundaries given by the QoS class.

1.7 Thesis Outline

This thesis is structured as follows. In the next chapter, we introduce and discuss previous work related to RTT predictions, embedding into metric spaces, building of overlay networks and overlay multicast services.

In chapter 3, we identify the computational overhead when computing the positions of landmarks in metric spaces of increasing dimensions. In the same chapter, we propose a heuristic which identifies the optimal number of dimensions for embedding RTT distances of landmarks. We also provide an algorithm which calculates a good starting point for the numerical function minimization used to calculate the positions of landmarks in a virtual space.

In chapter 4, we identify another problem inherent to systems which use function minimization to find the positions of end systems relative to a set of landmarks. The problem is that the sum of square errors function which needs to be minimized features multiple local minima. In the same chapter, we propose a solution to this problem and present the results of our evaluation.

In chapter 5, we introduce fisheye overlay, a new fully distributed algorithm and the corresponding communication protocol used to construct a topology-aware overlay network. This network is guaranteed to be an undirected connected graph, whose degree in each end system is bound by a constant number.

We put this overlay network to use in chapter 6. In this chapter we present a novel approach, named NetICE9, for embedding end systems into a k-dimensional Euclidean space. NetICE9 is similar to VIVALDI [16], since it is also a distributed simulation of a physical system. Unlike VIVALDI, NetICE9 is much more stable and results in better embedding of RTTs.

In chapter 7, we introduce an overlay network (NNCS Overlay) that guarantees success of greedy routing and at the same time is both simpler to construct and to maintain in a distributed manner compared to a Delaunay triangulation. At the same time, same as Delaunay Triangulation, NNCS guarantees the progress of greedy routing. We combine this overlay network with the fisheye overlay network presented in chapter 5 to obtain a system with very efficient greedy routing which forwards unicast messages through the overlay network. Also in chapter 7, we also present a system which performs spatially bound queries on every end system within a specified "volume" within the virtual space. We use this system to efficiently solve network optimization tasks such as finding QoS paths and locating services.

In the last chapter we summarize the contributions of this dissertation to the current state of research and give an overview on open issues as well as possible avenues for further research in this area.

Chapter 2

Related Work

In this chapter, we present work which is related to the topics of this dissertation.

2.1 Round-Trip Time Prediction Schemes

The main service provided by an RTT prediction scheme is to obtain a RTT for a give pair of end systems. Most RTT prediction schemes are based on the following idea: First, the end systems are embedded into a metric space; then, a distance function is used to predict RTTs. This has been a research topic for almost a decade. Numerous methods have been proposed to achieve this goal. To date, all methods proposed can be classified as either landmark-based or landmark-less. Landmark-based methods use so-called landmarks, a set of designated hosts within the network. Those landmarks usually measure RTTs amongst each other and use this information to position themselves within a virtual space. Then, all other hosts in the network measure their RTTs to the landmarks and, in the virtual space, position themselves relative to the landmarks according to the measured distances. Landmark-less approaches, on the other hand, usually perform a distributed simulation of the physical system. Such a simulation incrementally decreases the total error of the embedding of the RTTs in the virtual space and converges toward an optimal solution.

Both, landmark-based and landmark-less, approaches have their advantages and drawbacks. Landmark-based approaches are usually more stable than landmark-less ones, in the sense that the host positions in the virtual space are more stable. Unfortunately, landmark-based approaches usually have limited scalability and fault tolerance due to the fact that they heavily depend on landmarks as central components. Landmark-less approaches are usually completely distributed without any central components. This makes them very resilient to host failures and network outages. On the downside, landmark-less approaches are usually quite sensitive to violations of assumptions which underpin the simulation. For example, one of the most commonly made assumptions is that the triangle inequality holds for the measured RTTs. As shown in [21–23], however, this is not always the case in the Internet. Such violations lead to undesired effects such as the permanent oscillations of host positions in the simulation; or even worse, the rotation and translation of the whole system within the virtual space. Those effects lead to unstable host positions, making them useful

only for a limited time even if the underlying network remains unchanged.

As we will see in this section, not every RTT prediction scheme is based on the assumption that the triangle inequality holds. For example, IDMaps, which we describe below, has a completely different approach. Also, the matrix factorization method (also described in this Section) does not base its predictions on the use of metric spaces.

2.1.1 IDMaps

One of the pioneering ideas in the field of RTT prediction is IDMaps [24]. The authors of IDMaps propose the use of special network components, so-called "tracers". Tracers are dedicated network components which are installed in the core network of the ISP. Every tracer is aware of all other tracers in the Internet and periodically measures its RTTs to each of them. In IDMaps the estimated RTT between any two end systems a and b is defined to be the sum of the following distances (RTTs): $\hat{d}_{a,b} := d_{a,T_a} + d_{T_a,T_b} + d_{T_b,b}$, where d_{a,T_a} is the RTT between a and its nearest tracer (T_a), d_{T_a,T_b} is the distance between T_a and the tracer nearest to b (T_b) and $d_{T_b,b}$ is the distance between b and T_b.

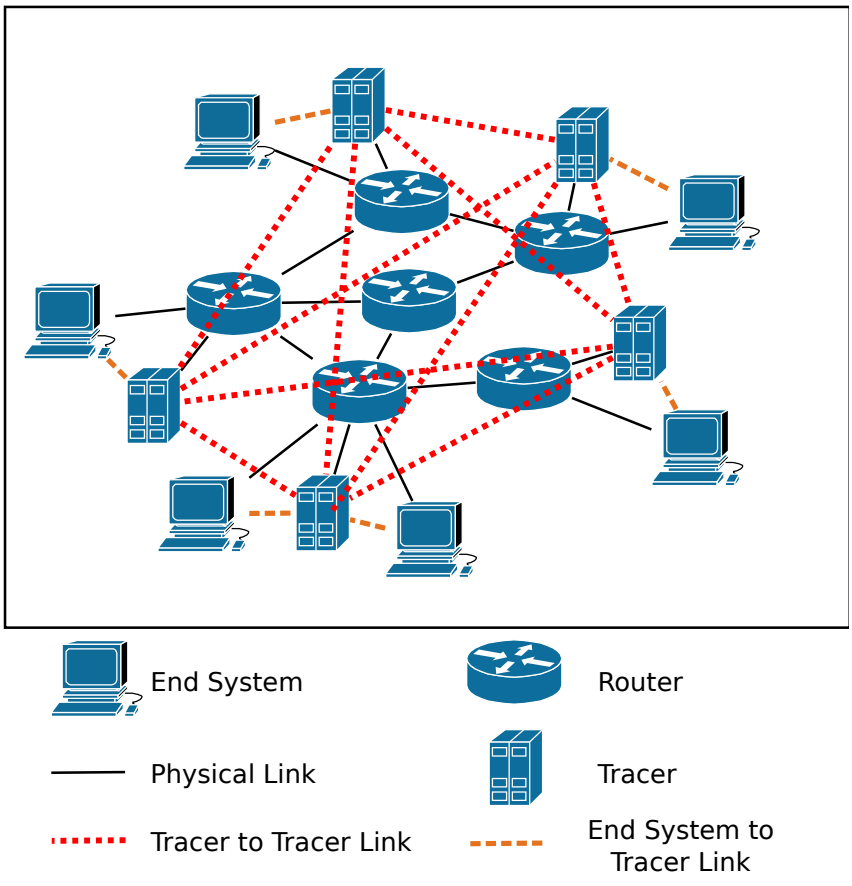

Figure 2.1: Architecture of IDMaps

The advantage of this approach is the number of tracers needed to cover the whole Internet. According to authors of IDMaps, this number can be kept low to such an extent that an IDMaps infrastructure in the Internet would be feasible.

However, this approach also has many disadvantages. The most severe disadvantage is the requirement to deploy an infrastructure (tracers) in the core networks of the internet service providers. Furthermore, the obtained RTT predictions tend to be significantly less precise than those yielded by other approaches such as GNP.

2.1.2 Global Network Positioning (GNP)

The historically first and the most promising RTT prediction scheme which uses the embedding of RTTs into metric spaces is Global Network Positioning (GNP) [15, 17]. In GNP, m end systems are chosen as landmarks (denoted by $\mathcal{L}_1 \ldots \mathcal{L}_m$). All other end systems (denoted by $\mathcal{H}$) in the Internet measure their RTTs (denoted by $\hat{d}_{\mathcal{H},\mathcal{L}_1}, \ldots, \hat{d}_{\mathcal{H},\mathcal{L}_m}$) to the landmarks and use this information to determine their position in a Euclidean virtual space $\mathcal{S}$. The coordinates of an end system in $\mathcal{S}$ (denoted by $\mathcal{C}_\mathcal{H} := (c^1_\mathcal{H}, \ldots, c^d_\mathcal{H})$) are determined by using function minimization. The function minimized is the sum of square errors between the measured and the predicted distances (RTTs) f_e. In the case of GNP, f_e is defined as:

$$f_e(\mathcal{C}^\mathcal{S}_\mathcal{H}) := \sum_{i=1}^{m} (d_\mathcal{S}(\mathcal{C}^\mathcal{S}_\mathcal{H}, \mathcal{C}^\mathcal{S}_{\mathcal{L}_i}) - \hat{d}_{\mathcal{H}\mathcal{L}_i})^2. \tag{2.1}$$

The function minimization is performed using the Downhill Simplex [20] method.

For calculating $d_\mathcal{S}(\mathcal{C}_\mathcal{H}, \mathcal{C}_{\mathcal{L}_i})$, the positions of the landmarks in $\mathcal{S}$ ($\mathcal{C}_{\mathcal{L}_1} \ldots \mathcal{C}_{\mathcal{L}_m}$) must be known. In the case of GNP, the coordinates of the landmarks are computed by minimizing the following error function:

$$f_e(\mathcal{C}^\mathcal{S}_{\mathcal{L}_1}, \ldots, \mathcal{C}^\mathcal{S}_{\mathcal{L}_m}) := \sum_{i,j \in \{1 \ldots m\}, j>i} (d_\mathcal{S}(\mathcal{C}^\mathcal{S}_{\mathcal{L}_i}, \mathcal{C}^\mathcal{S}_{\mathcal{L}_j}) - \hat{d}_{\mathcal{L}_i\mathcal{L}_j})^2. \tag{2.2}$$

In the same paper, the authors of GNP propose the following alternative normalized error function:

$$f'_e(\mathcal{C}^\mathcal{S}_{\mathcal{L}_1}, \ldots, \mathcal{C}^\mathcal{S}_{\mathcal{L}_m}) := \sum_{i,j \in \{1 \ldots m\}, j>i} \left(\frac{d_\mathcal{S}(\mathcal{C}_{\mathcal{L}_i}, \mathcal{C}_{\mathcal{L}_j}) - \hat{d}_{\mathcal{L}_i\mathcal{L}_j}}{\hat{d}_{\mathcal{L}_i\mathcal{L}_j}} \right)^2.$$

The reason for proposing f'_e as an alternative to f_e is that it yields better results when the relative error performance metric defined as:

$$\left| \frac{d_\mathcal{S}(\mathcal{C}_{\mathcal{L}_i}, \mathcal{C}_{\mathcal{L}_j}) - \hat{d}_{\mathcal{L}_i\mathcal{L}_j}}{\min\left(d_\mathcal{S}(\mathcal{C}_{\mathcal{L}_i}, \mathcal{C}_{\mathcal{L}_j}), \hat{d}_{\mathcal{L}_i\mathcal{L}_j}\right)} \right|$$

is used to evaluate the performance of the RTT predictions.

Although GNP yields very good RTT predictions, it also has few disadvantages. On of the disadvantages of GNP is the use of central components – the landmarks. In GNP, landmarks have to be highly available and cope with high amounts of traffic because all end systems perform their RTT measurements relative to the landmarks. If the number of end system is low, this is not a problem. However, for large numbers of end systems, this can lead to a high network load.

2.1.3 Network Positioning System (NPS)

In order to make GNP more scalable, its authors proposed a more distributed system which they named Network Positioning System (NPS) [17]. Like GNP, NPS has a initial set of landmarks. Unlike GNP, NPS introduces several layers of landmarks. Initial landmarks are on the first layer, landmarks which position themselves relative to the first layer are called second layer landmarks, etc. Each layer of landmarks contains more landmarks than the previous one and uses only landmarks from lower layers to determine the position of its landmarks. End systems which are not used as landmarks can also be considered to be the last layer of landmarks.

This infrastructure design is much more scalable, since temporary outages of individual landmarks do not stop the service. Also, the load of RTT measurements is not focused on just a few landmarks.

2.1.4 VIVALDI

In principle, VIVALDI [16] is a distributed simulation of a physical system. This physical system consists of hosts positioned in a virtual k-dimensional Euclidean space. Every pair of hosts is connected by a virtual spring. The force stored in this spring is proportional to the difference between the distance between two hosts within the virtual space and the actual RTT which was measured between them. Each host periodically corrects its position within the virtual space by randomly choosing a neighbor, measuring the RTT to it and adjusting its position by either moving closer to or further away from the chosen neighbor to lessen the force stored by the virtual spring; i.e. the host adjusts its position to reduce the embedding error. In order to reduce oscillations (moving back and forth in the virtual space without changing the position on average) of the whole system, the authors of VIVALDI proposed that each step of an optimization (moving closer to or away from a neighbor to relax the force of the spring) is not performed fully. Instead, in each step, the spring is relaxed only partially. As an estimate of how much the spring should be relaxed, the authors of VIVALDI suggested using estimates of the embedding error of the host performing the optimization and its neighbor. This estimate is calculated at each step of the optimization The goal is to prefer relaxing the springs to those neighbors which have better estimates of their position.

In addition to a host's position in the virtual space, the authors of VIVALDI also recommended the introduction of a non-negative component, a so-called "height vector". This component represents the non-metric part of the host position. The purpose of the height

vector is to further increase the precision of the prediction by approximately representing the distance of the host to the backbone of the Internet, which VIVALDI models as metric.

Due to its simplicity (no sophisticated function minimization is needed) and because it does not need any infrastructure (landmarks or similar) VIVALDI has become very popular for various peer-to-peer applications [25–27]. In chapter 6 we propose improvements for VIVALDI, which will increase its stability and improve its prediction precision.

2.1.5 Internet Coordinate System (ICS)

The approach used by the Internet Coordinate System (ICS) [28] is similar to the one used by GNP. Like GNP, ICS uses a set of landmarks whose positions within the virtual space are determined by the RTTs measured between them. Unlike GNP, ICS does not use function minimization to position the landmarks. Instead, ICS does a PCA (principal component analysis) of the data in order to obtain a transformation matrix. This matrix can then be used to transform (by simple matrix multiplication) the RTTs measured from a host to each of the landmarks into that particular host's position.

The advantage of this approach is clearly on the computational side. Performing PCA and multiplying matrices is far less demanding on computational resources than minimizing functions by means of some numerical method, as is the case with GNP.

2.1.6 Practical Network Coordinates (PIC)

Practical Network Coordinates (PIC) [29] is a RTT prediction scheme similar to GNP. Exactly as in GNP, every end system joining PIC uses l end systems as landmarks. Unlike GNP, PIC does not always use the same landmarks. Instead, PIC chooses a hybrid of nearest neighbors and random choice of end systems that are already in the system.

In simulations, the authors of PIC have shown that this strategy yields a similar performance in terms of prediction accuracy as GNP. At the same time, PIC avoids drawbacks imposed by using landmark end systems such as single point of failure and excessive load on landmarks.

2.1.7 Lighthouses

Lighthouses [30] tackles the scalability issue of GNP. In Lighthouses, every end system arbitrarily selects already positioned end systems and measures its RTTs to them. This RTT information is then used to construct a new (local) coordinate system in which the end system positions itself. In a next step, a transformation matrix is computed which transforms the positions of that new local coordinate system into a global one. Simulations have shown that this approach yields results of similar accuracy as GNP, while avoiding the scalability issues of GNP.

2.1.8 BIGBANG

Another landmark-based approach is BIGBANG [14]. BIGBANG uses a simulation of a physical system to achieve a computationally more efficient embedding for both, landmarks and hosts. Basically, BIGBANG does not differ much from GNP, only the function minimization strategy is more computationally efficient. This approach was inspired by the big bang simulations used in astrophysics.

Another interesting idea is the follow-up work on BIGBANG [31]. Here, the authors propose the use of a hyperbolic metric space for embedding rather than a Euclidean one. In this approach, authors propose embedding RTTs into a Poincaré Disc. This yields far better results, since graphs which represent the Internet are more easily embedded into a hyperbolic metric space than into a Euclidean space.

2.1.9 IDES

IDES, the Internet distance estimation service [32], provides an alternative approach for RTT predictions. In this approach, rather than embedding end systems into a metric space, the authors propose one of the matrix factorization methods such as singular value decomposition (SVD) or non-negative matrix decomposition (NMF).

In order to have a prediction service, the authors of IDES propose a landmark structure similar to the ones in GNP or ICS. The next step, however, differs. To obtain some kind of "coordinates" for each end system, the landmark data matrix is factorized and a linear equation system for all the other end systems is solved. Those "coordinates" are then represented by two (incoming and outgoing) vectors. An estimate of the communication delay is then obtained by using the dot product of a matched pair of those vectors. This way, IDES can estimate the non-symmetric delay information. Also, IDES does not require the delay information to fulfill the triangle inequality.

On the down side, since IDES does not determine positions in a metric space, this approach does not allow us to use position-based algorithms to control routing-decisions in the overlay network. Also, this approach suffers from similar limitations as other landmark-based approaches which require landmark end systems to be available.

2.1.10 Meridian

Meridian [33] is an overlay network designed to find nearby RTT-related services. Similar to our approach, which we will introduce shortly, Meridian gives each of its end systems a fisheye view of its neighbors. The fisheye of each end system consists of concentric rings with increasing radii. Each of the rings contains a constant number of neighbors k. The geographic diversity of the chosen neighbors within one ring is ensured by maximizing the k-polytope formed by the neighbors. Since Meridian does not guarantee the formation of a connected graph, it is not suitable for the purpose of unicast or multicast routing. Instead, the purpose of Meridian is to provide an efficient neighbor finding service which optimizes the communication delay. The number of hops of a query is guaranteed to scale

logarithmically $O(\log n)$ relative to the number of end systems participating in the Meridian network.

2.1.11 Binning

Binning [34] is a way of grouping end systems which are close to each other in terms of RTT. They are grouped within bins (specific areas of the network). Similar to GNP, Binning uses a set of landmarks (designated end systems), to which every end system measures its RTTs. Unlike GNP, Binning does not try to embed the RTTs into a metric space. Instead, every end system uses the obtained RTTs to determine which partition of the network it belongs to; i.e. where it is located – its bin. The bin is determined by sorting landmarks in ascending order according to the measured RTTs. For example, if the end system h has measured RTTs $(20, 15, 40)$ to landmarks (L_1, L_2, L_3) then its bin is L_3, L_1, L_2.

If two end systems are located in the same bin, then they are located in the same region of the network. Also, the similarity of two bins can be assessed by comparing the length of the matching prefix of the bins. The longer the matching prefix, the more similar are the two bins, i.e. the nearer to each other are they located.

The authors propose to use Binning information to construct a topology-aware neighbor selection mechanism for an overlay network that contains both near and far neighbors. They suggest to enhance the CAN [8] overlay network with Binning in order to obtain a topology-aware CAN overlay.

2.2 Overlay Networks

2.2.1 Chord

Chord [6] is one of the first structured overlay networks. Chord consists of a one-dimensional key space of the size 2^m (m is usually 128) arranged as a ring. Each end system participating in a Chord overlay network randomly chooses its own Chord address (a key address). Each end system keeps track of its direct neighbors, i.e. those two end systems whose key addresses are closest to (a bit lower and a bit higher than) its own. Additionally, every end system keeps a so-called finger table, i.e. a routing table. Similar to other approaches such as Pastry [7], the finger table of Chord, due to its segmentation, represents a fisheye view of the Chord overlay network from the viewpoint of the end system. Routing in the chord overlay network is done by choosing the finger from the finger table that is numerically closest to the destination address. Since every end system has a fisheye view of the overlay network, the routing process is guaranteed to succeed in $\log_m n$ steps, where n is the number of end systems participating in the overlay network.

The main drawback of Chord is that it is completely unaware of the topology of the underlying network. Both, the choice of Chord address and the choice of neighbors in the finger table is totally topology-agnostic. As a result, Chord performs rather poorly when the total RTT of the routing path is considered.

2.2.2 CAN

CAN [8] is a structured overlay network providing services such as a distributed hash table (DHT) and overlay Multicast service. The basic principle of CAN is the following. In the DHT of CAN, a point in a d-dimensional torus (wrap around) virtual space is used as a key for the stored object. Each end system participating in CAN is responsible for one portion (volume) of the virtual space. This means that an end system stores the information of all objects with keys that are within that portion of the virtual space for which it is responsible.

Routing in CAN is relatively simple. In order to find the end system that is responsible for storing a specific object with a defined key K, we may start the lookup at any end system within CAN overlay network. The next hop of the routing is always the neighbor of that end system which is nearest to the position of K (greedy routing).

When a new end system joins a CAN overlay network, it randomly chooses a point in the virtual space, finds the end system responsible for the that portion of the virtual space and then divides that virtual space in two equal parts. Subsequently, one part of the virtual space becomes the responsibility of the new end system, while the other part remains the responsibility of the end system that was responsible for the original portion of the virtual space.

CAN itself uses positions in a virtual space of end systems to distribute the load of storing objects and to facilitate their efficient lookup. It uses position-based algorithms to route the requests. Unfortunately, the positions of the end systems are randomly chosen, which means that CAN is not topology-aware.

2.2.3 Pastry

Another interesting topology-aware overlay protocol is Pastry [7]. Pastry is a general-purpose overlay network routing protocol. Each node in Pastry has an unique (usually randomly chosen) ID. Routing in Pastry is done in a Plaxton-like [35] way using a prefix routing table. Like in many other overlay/p2p routing protocols, this results in a fisheye view of the address space. The feature that makes Pastry unique among other prefix-based routing schemes is the way the routing table is built. If more than one candidate exists for an entry in the prefix table, Pastry chooses the one with smallest RTT. Consequently, Pastry optimizes not only the routing in the overlay, but also in the underlying network.

2.3 Overlay Multicast

2.3.1 SCRIBE

Scribe [36] is an application-layer overlay network built on top of a Pastry [7] structured overlay network protocol.

Scribe supports multiple multicast groups in one Pastry ring. Every group is mapped to one Pastry address. The end system whose Pastry node address is nearest to the group address is chosen to be the root of the multicast group. Every end system that is interested in receiving multicast traffic for a group sends a join message in the direction of the group

address through the Pastry overlay network. On its path, the join message updates the routing tables of the end systems it traverses. Thus, a tree consisting of reverse paths from leaves to root is constructed. Every multicast message is first sent via unicast to the root node of the group. The root node then floods the message down the multicast tree of the group.

2.3.2 NICE

NICE [37] is a topology-aware overlay multicast protocol. NICE consists of clusters of end systems. The size of the clusters is between k to $3k - 1$ end systems (k is a small number). In every cluster there is a cluster leader, which is the graph-theoretic center (using RTT as a distance measure) of the cluster. All clusters are layered, with the lowest layer L_0 containing all clusters and hence all end systems. The next layer (L_1) contains clusters of L_0 cluster leaders. Each subsequent layer consists of clusters of cluster leaders from the preceding layers. In the final layer L_n, there is only one cluster.

When a new end system joins a NICE overlay network, it first measures the RTT to every end system in the final layer L_n. Using the measured RTTs, the end system with smallest distance (RTT) is chosen. This end system is a cluster leader for a cluster in the lower layer (L_{n-1}). Now, the procedure is repeated with this cluster in order to determine the next cluster one layer lower. When the lowest layer (L_0) is reached, the end system joins that cluster. If the new cluster is larger than the predefined maximal cluster size ($3k - 1$), this cluster is split in two new clusters, with two new cluster leaders.

2.3.3 NARADA

One of the first proposed topology-aware application-layer multicast (ALM) is NARADA [38]. NARADA is designed for relatively small groups of end systems. The approach of NARADA is to build a connected graph of end systems which the authors termed “mesh”. The mesh is a graph that is constantly refined according to a utility function which optimizes the utility gain (reduction of RTT) for the other neighbors. For each sender in NARADA, a distribution tree, which is a sub graph of the mesh, is constructed. The construction of the distribution tree is done using a DVMRP [39]-like protocol.

The main issue of NARADA is its scalability. Since the protocol was designed for relatively small groups, each end system knows the complete state of the overlay network. This approach does not scale, since the memory available on each end system is the limiting factor for the size of the overlay network.

2.4 Round-Trip Time Measurements

2.4.1 Planet-Lab

Planet-Lab [40] is a large distributed test bed of computers which are located all over the Internet. The idea behind Planet-Lab is to have a platform on which to test networking

protocols, especially prototype peer-to-peer protocols. Planet-Lab nodes are distributed all over the world and are usually hosted by research institutions such as universities. This diversity of location of Planet-Lab nodes gives researchers the unique opportunity to test their prototypes in an environment which is quite similar to the conditions the application would encounter on the Internet.

Most of the experiments run in the Planet-Lab are short term experiments with usual runtimes of a few weeks or months. There are some experiments that have been running for a very long time. One of those experiments is the All-Sites-Ping experiment [41]. The All-Sites-Ping experiment runs on all nodes of the Planet-Lab. Every 30 minutes, the All-Sites-Ping experiment performs an ICMP-Echo/ICMP-Echo-Reply measurement (ping) to all other nodes in the Planet-Lab. The results of of those measurements are then aggregated and presented as an RTT distance matrix of the whole Planet-Lab.

We used this data to perform our simulations. In all the evaluations described in this dissertation, we used the data gathered from the All-Sites-Ping experiment for our network model. Running with the assumption that the delay is the same in both directions, in our network model, we delayed every packet from end system A to end system B by half of the RTT measured in Planet-Lab between A and B.

2.4.2 King

King [42] is an interesting approach used to estimate the RTT between two arbitrary end systems in the Internet. Whenever an end system C needs an RTT prediction between end systems A and B it can use the King method to obtain the prediction. King predicts RTTs by performing a recursive DNS query between DNS servers located near A and B respectively. Those DNS servers, denoted by D_A and D_B, are found by a reverse DNS lookup for the addresses of A and B. Once the servers are found, C sends a recursive DNS query to D_A inquiring about DNS information which is located at D_B. In order to respond, D_A queries D_B and sends the result to C. The total time which has elapsed between the moment when C sent its recursive DNS query to D_A and the time when C receives the the response from D_A contains the sum of RTTs: $RTT(C, D_A) + RTT(D_A, D_B)$. To obtain the RTT between D_A and D_B, all the end system C has to do is measure the RTT from itself to D_A and subtract this time from the total time it took to process the DNS request (for details see Figure 2.2). King uses the measured RTT between D_A and D_B as an estimate for the RTT between A and B.

By doing so, King makes a few assumptions. The biggest assumption is that DNS servers are located topologically close to the end systems they are responsible for. Also, King assumes that the time it takes DNS servers to process requests is negligible compared to the measured RTTs.

The authors of King have shown in their work that in most of the cases in the Internet both assumptions are correct. Hence, in chapters 5, 6 and 7, we used data which was obtained with the King method in the network model of our simulations.

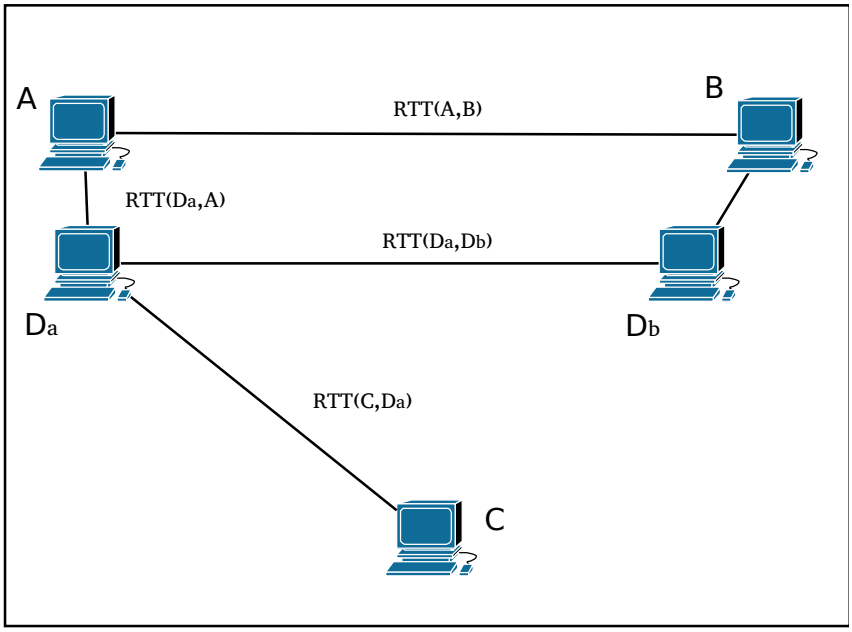

Figure 2.2: KING: Using Nearby DNS Server to Obtain RTT Prediction

Chapter 3

Optimizing Dimensionality and Accelerating Landmark Positioning for Coordinate-Based Round-Trip Time Predictions

3.1 Introduction

In recent years, numerous proposals have been made for RTT prediction schemes based on coordinates [15–17, 28, 29, 43, 44]. Basically, there are two types of coordinates-based RTT prediction schemes: landmark-based and landmark-less. Landmark-based RTT prediction schemes determine the coordinates of a few fixed reference points (so-called landmarks) and use them to determine the positions of all other end systems. Some of the landmark-based schemes use function minimization to position the landmark end systems [15, 17, 29]. Others [28, 44] use principal component analysis (PCA) to reduce the dimensionality of the landmark RTT measurements and to position the end systems. Landmark-less RTT prediction schemes such as VIVALDI [16] or S-VIVALDI [43] are based on a distributed simulation of physical systems to iteratively reduce the overall error of the end system embedding in an Euclidean space.

Whatever landmark positioning strategy is used, it is crucial that the landmark coordinates are as accurate as possible, since the coordinates of all other end systems in the Internet are calculated relative to them. An error in the position of one landmark decreases the precision of the predicted RTTs for almost every end system in the Internet.

There are numerous problems with the way the positioning of landmarks is done in GNP. The most severe one is the computational overhead of the function minimization. The function minimization used for GNP (Downhill Simplex) is very computation-intensive for several reasons. There is the exponential growth of the problem space with each additional variable, which is further complicated by the lack of a "good" starting point (a point in the vicinity of the optimal solution). Furthermore, there are an infinite number of possible landmark positions. This slows down the convergence because there are many solutions toward which the Downhill Simplex may converge.

In this chapter, we describe how we improved landmark positioning in GNP as published in [45]. We improved the positioning of landmarks in GNP by reducing the number of possible solutions to a finite number. Then, we determined the optimal number of dimensions for the space in which the landmarks can be positioned. And, finally, we provided a good starting point for the function minimization.

The structure of this chapter is the following: In the next Section we show on an example that there is a practical problem with the computational complexity for dertermining positions of landmars in GNP. In Section 3.3 we provide a formal description for the problem, i.e. finding landmark coordinates. In Section 3.4, we discuss how the infinite number of solutions can be reduced to a finite number and provide an explicit method for constructing a simplex for the given edge lengths. In Section 3.5, we describe the possible violations of the properties of a k-dimensional metric space (the Simplex Inequality) and how those can be detected (using Cayley-Menger determinants). In Section 3.6, we describe algorithms that utilize the results from Sections 3.4 and 3.5 to calculate the optimal number of dimensions for representing landmark coordinates, and to compute a good starting point for finding them. In Section 3.7, we evaluate the proposed algorithm by comparing its accuracy, performance and number of dimensions with the landmark positioning scheme proposed in GNP.

3.2 Complexity of Function Minimization

The first question to ask is whether there is a scaling problem for GNP's landmark positioning regarding the number of dimensions used. As we have mentioned, GNP proposed positioning landmarks by minimizing a least squares objective function f_e defined in (2.2). At the first glance, computing f_e does not depend on the number of dimensions k. If we consider the definition of the Euclidean distance between two landmarks $d_{\mathcal{S}}$ (3.1), it is clear that the computational overhead of $d_{\mathcal{S}}$ is linear ($d_{\mathcal{S}} \in O(d)$), which also means that $f_e \in O(d)$.

$$d_{\mathcal{S}}(\mathcal{C}_{\mathcal{L}_i}, \mathcal{C}_{\mathcal{L}_j}) := \sqrt{\sum_{m=1}^{k} (\mathcal{C}^m_{\mathcal{L}_i} - \mathcal{C}^m_{\mathcal{L}_j})^2} \tag{3.1}$$

To determine the computational complexity of the function minimization we still have to determine the number of evaluations of f_e needed to perform the function minimization. An analytical approach to find this number is impractical due to the fact that the number of iterations needed heavily depends on the starting point used to perform the minimization. To still be able to roughly determine the computational overhead of the function minimization, we have gathered empirical data and performed a statistical analysis of it.

For the experiment, we have randomly chosen 20 hosts from the Planet-Lab testbed as landmarks. For those 20 landmarks we retrieved the full RTT measurement matrix from the PlanetLab all-sites-pings experiment [41]. For this distance matrix we used the Downhill Simplex function minimization as proposed in GNP to determine the landmark positions with the number of dimensions k varying between 1 and 19, which is the theoretical maxi-

mum of dimensions with 20 landmarks. For each number of dimensions we have performed 50 function minimizations with different randomly chosen starting points. For each function minimization we have recorded the number of evaluations of the objective function f_e performed to obtain the landmark coordinates.

The results of this experiment are summarized in Fig. 3.1. This Figure contains the median and the 90% confidence interval of the number of function evaluations performed for each number of dimensions. As we can see, the number of function evaluations performed

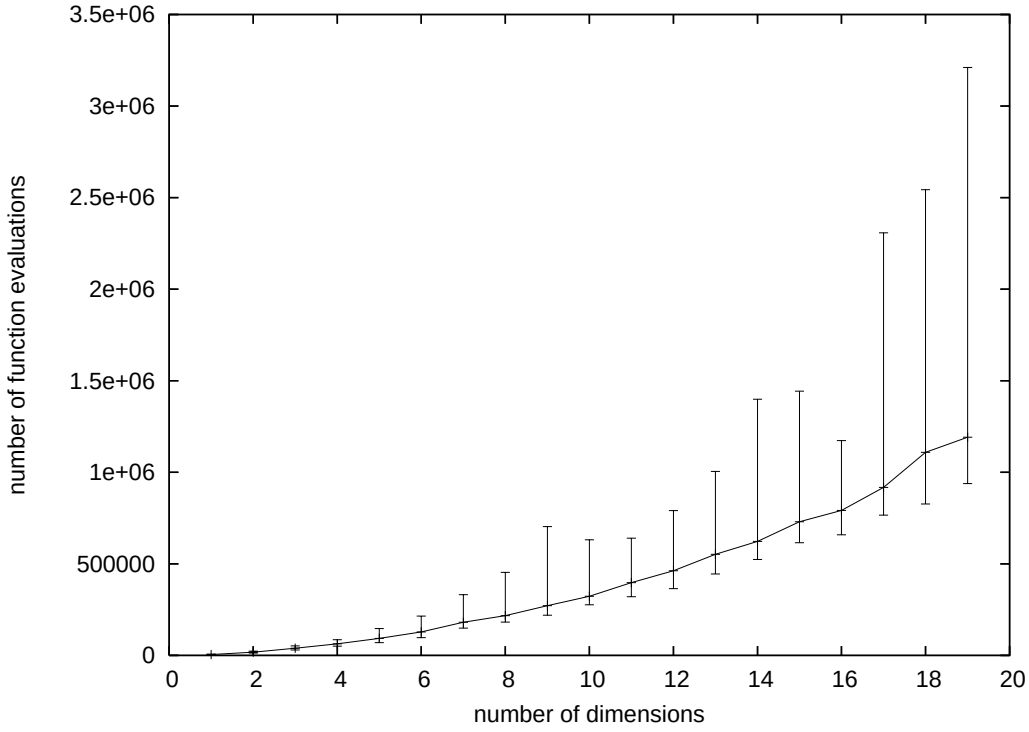

Figure 3.1: Number of evaluations of the objective function used to position landmarks

is increasing more than linearly with the number of dimensions used. Our assumption judging on the shape of the curve is that it increases quadratically. Our assumption is confirmed by Fig. 3.2 where we have represented the square root of the number of performed function evaluations depending on the number of dimensions. The median values in this Figure increase linearly, meaning that the original function is increasing quadratically.

This result allows us to state that the number of function evaluations is within $O(k^2)$. As shown above, each function evaluation has a linear computational overhead regarding the number of dimensions. Therefore, we can state that the total computational complexity of an average positioning of landmarks depending on the number of dimensions k is cubic ($O(k^3)$).

Figure 3.3 shows the average CPU time needed (in seconds) measured in our experiment to find landmark positions depending on the number of dimensions. This Figure clearly

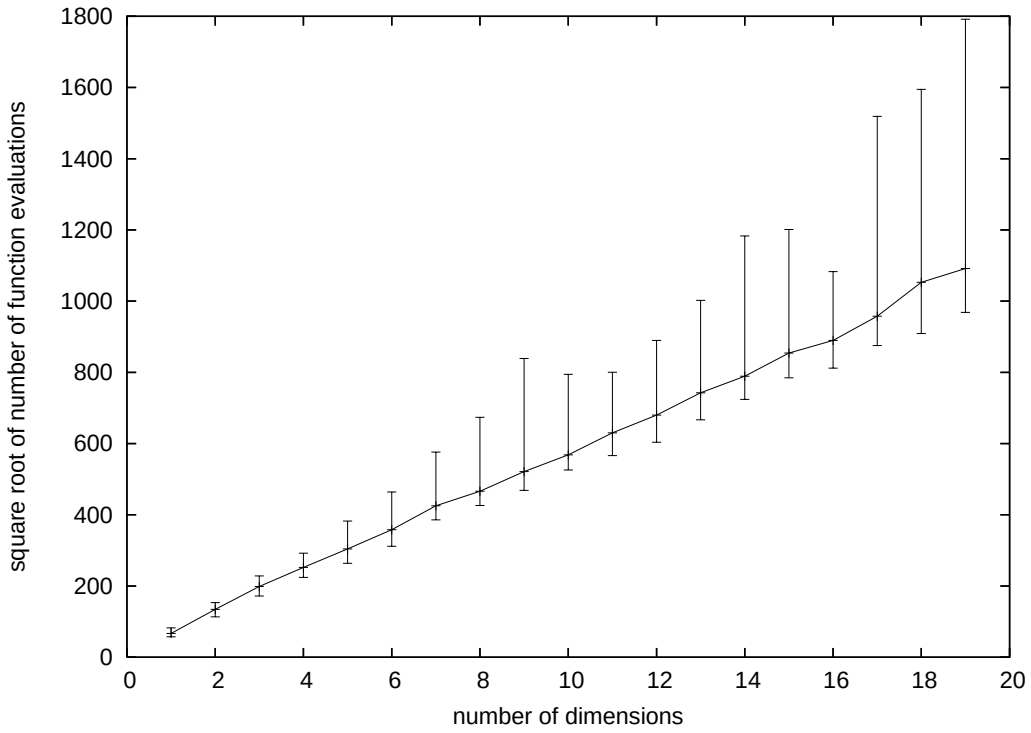

Figure 3.2: Square root of the number of evaluations of the objective function used to position landmarks

demonstrates that reducing the number of dimensions can dramatically accelerate the time needed to position the landmarks. For example positioning 20 landmarks in 19 dimensions takes in average 147.795 seconds CPU time. Finding landmark positions for the same data in 6 dimensions takes only 6.045 seconds. In our opinion, results of this experiment justify the need for reducing the number of dimensions used for positioning landmarks.

3.3 Landmark Coordinates: The Problem Defined

In GNP we have a set of m landmarks $(\mathcal{L}_1, \ldots, \mathcal{L}_m)$ and a complete set of measured distances between the given landmarks $\{d_{\mathcal{L}_i,\mathcal{L}_j} : i, j \in \{1, \ldots, m\}\}$, which can be represented as the following distance matrix:

$$\begin{pmatrix} 0 & d_{\mathcal{L}_1,\mathcal{L}_2} & \cdots & d_{\mathcal{L}_1,\mathcal{L}_{m-1}} & d_{\mathcal{L}_1,\mathcal{L}_m} \\ d_{\mathcal{L}_2,\mathcal{L}_1} & 0 & \cdots & d_{\mathcal{L}_2,\mathcal{L}_{m-1}} & d_{\mathcal{L}_2,\mathcal{L}_m} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d_{\mathcal{L}_{m-1},\mathcal{L}_1} & d_{\mathcal{L}_{m-1},\mathcal{L}_2} & \cdots & 0 & d_{\mathcal{L}_{m-1},\mathcal{L}_m} \\ d_{\mathcal{L}_m,\mathcal{L}_1} & d_{\mathcal{L}_m,\mathcal{L}_2} & \cdots & d_{\mathcal{L}_m,\mathcal{L}_{m-1}} & 0 \end{pmatrix}$$

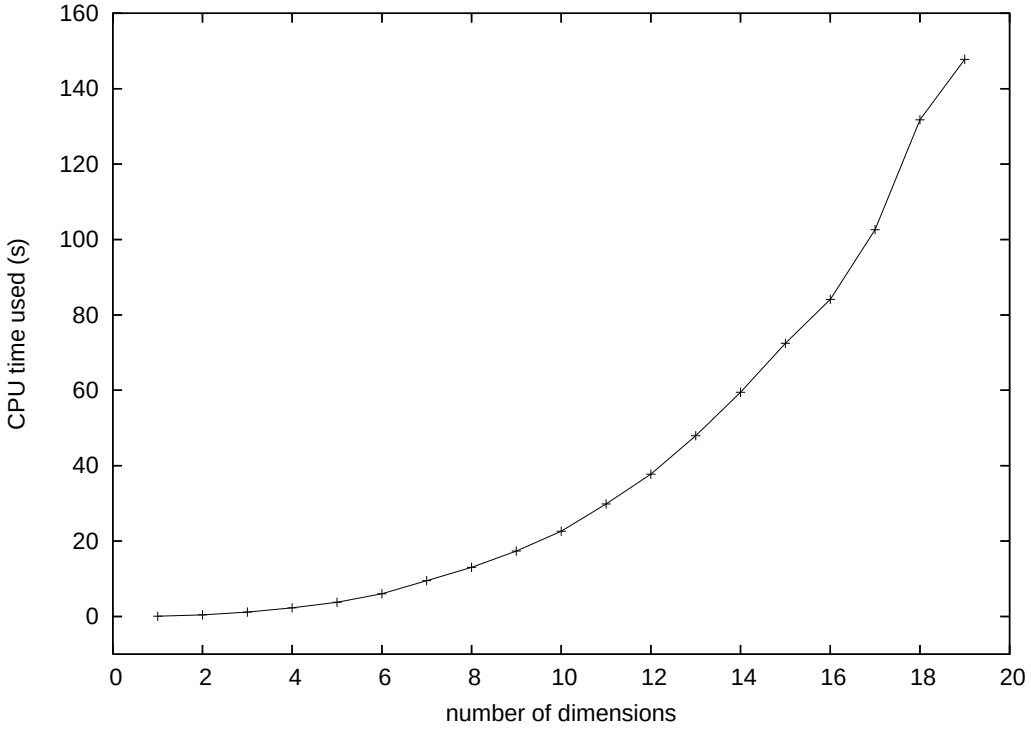

Figure 3.3: Average CPU time in a 3 GHz Pentium-D CPU (in seconds) for computing landmark positions depending on the number of dimensions

We assume that the distance matrix is symmetric about the main diagonal, meaning that $d_{\mathcal{L}_i,\mathcal{L}_j} = d_{\mathcal{L}_j,\mathcal{L}_i}$.

For this input, we are interested in finding the landmark coordinates $(\mathcal{C}^{\mathcal{S}}_{\mathcal{L}_1}, \ldots, \mathcal{C}^{\mathcal{S}}_{\mathcal{L}_m})$ in the k-dimensional Euclidean space $\mathcal{S} := (\mathbf{R}^d, d_{\mathcal{S}})$. If the distance measurements were error-free, we would be able to construct an $(m-1)$-simplex, which is the simplest regular figure that can be constructed using m points in an $(m-1)$-dimensional space, with points as the landmarks and side lengths equal to the measured distances between them. It is obvious that all solutions to our landmark coordinates problem, based on an ideal input, can be obtained by applying isometric transformations to one solution. Isometric transformations are transformations that do not change the distances between each pair of points such as rotation, translation and reflection. This means that in an $(m-1)$-dimensional space there is an infinite number of solutions that represent the positions of a simplex. For example, Figure 3.4 shows some of the landmark positioning solutions in two dimensions for the following distance matrix:

$$\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & \sqrt{2} \\ 1 & \sqrt{2} & 0 \end{pmatrix}$$

As we can see, any one of the triangles in Figure 3.4 is a solution for the landmark po-

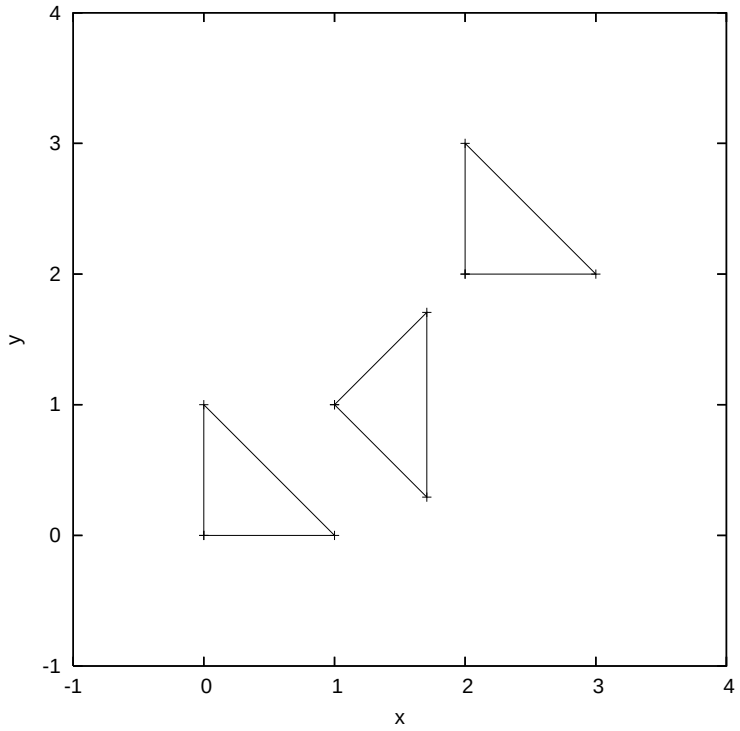

Figure 3.4: Example of isometrically equivalent solutions for the coordinates of a 2-simplex

sitioning problem. In other words, the two isometric operations rotation and translation generate an infinite number of solutions. This is one of the main reasons why the function minimization used in GNP to determine landmark coordinates converges very slowly and is non-deterministic.

A higher number of dimensions also increases the complexity of the problem. In addition to rotation and translation, we may now find simplices of several different shapes but which still have the same distances between vertices. It is small comfort then that the number of such different shapes is finite. In the following Section we will explain how the number of possible solutions can be reduced to a finite number and how one of those solutions can be explicitly computed.

3.4 Constructing a Simplex

If we have a distance matrix as described in Section 3.3 and assuming that it is possible to construct an $(m-1)$-simplex from the input (for a method to verify this assumption see Section 3.5), reconstructing one such simplex is equivalent to finding one solution for the

following equation system:

$$\begin{aligned}
\sum_{i=1}^{m-1}(\mathcal{C}^i_{\mathcal{L}_1} - \mathcal{C}^i_{\mathcal{L}_2})^2 &= (d_{\mathcal{L}_1,\mathcal{L}_2})^2 \\
\sum_{i=1}^{m-1}(c^i_{\mathcal{L}_1} - c^i_{\mathcal{L}_3})^2 &= (d_{\mathcal{L}_1,\mathcal{L}_3})^2 \\
&\vdots \\
\sum_{i=1}^{m-1}(\mathcal{C}^i_{\mathcal{L}_1} - \mathcal{C}^i_{\mathcal{L}_m})^2 &= (d_{\mathcal{L}_1,\mathcal{L}_m})^2 \\
\sum_{i=1}^{m-1}(\mathcal{C}^i_{\mathcal{L}_2} - \mathcal{C}^i_{\mathcal{L}_3})^2 &= (d_{\mathcal{L}_2,\mathcal{L}_3})^2 \\
&\vdots \\
\sum_{i=1}^{m-1}(\mathcal{C}^i_{\mathcal{L}_{m-1}} - \mathcal{C}^i_{\mathcal{L}_m})^2 &= (d_{\mathcal{L}_{m-1},\mathcal{L}_m})^2
\end{aligned} \tag{3.2}$$

This equation system requires that all distances between the vertices of the simplex are exactly the same as the measured distances. Under the assumption that there exists at least one solution to this equation system, then there is an infinite number of other solutions (as we have shown in Section 3.3) for the same equation system.

3.4.1 Eliminating Translation and Rotation

To reduce the number of solutions to a finite number, we have to eliminate the isometric operations that yield an infinite number of solutions. Those operations are rotation and translation.

Translation could be eliminated by requiring one landmark to be in a fixed position – for example we could require that the first landmark ($\mathcal{L}_1$) must be positioned in the origin of $\mathcal{S}$: $\mathcal{C}^{\mathcal{S}}_{\mathcal{L}_1} = (0, \ldots, 0)$. Unfortunately, it is not enough to eliminate translation. Even if we restrain the first landmark to a fixed position, we can still obtain an infinite number of solutions by rotating the simplex about an axis which runs trough that fixed point.

To eliminate rotation we require the second landmark ($\mathcal{L}_2$) to be located on the first axis of the space $\mathcal{S}$, i.e. only the first coordinate of the landmark is variable, the rest of them are zero: $\mathcal{C}^2_{\mathcal{L}_2} = 0, \ldots, \mathcal{C}^{m-1}_{\mathcal{L}_2} = 0$. The coordinates of the third landmark ($\mathcal{L}_3$) should be located in the plane defined by the first two axes of $\mathcal{S}$, i.e. $\mathcal{C}^3_{\mathcal{L}_3} = 0, \ldots, \mathcal{C}^{m-1}_{\mathcal{L}_3} = 0$. Each subsequent landmark is allowed one more dimension. The last landmark has no constraints for its coordinates, since we have m equations and with every new equation we introduce one more dimension of freedom.

By constraining the landmark coordinates we eliminate any possibility for rotation. Restraining the position of the second landmark to the first axis prevents the simplex from

rotating around any other axis. Restricting the third landmark to a plane defined by the first two axes eliminates rotation around the first axis, etc. This means, if we have the coordinates of landmarks $(\mathcal{L}_1, \ldots, \mathcal{L}_m)$ represented as:

$$\begin{pmatrix} \mathcal{C}^1_{\mathcal{L}_1} & \mathcal{C}^2_{\mathcal{L}_1} & \ldots & \mathcal{C}^{m-2}_{\mathcal{L}_1} & \mathcal{C}^{m-1}_{\mathcal{L}_1} \\ \mathcal{C}^1_{\mathcal{L}_2} & \mathcal{C}^2_{\mathcal{L}_2} & \ldots & \mathcal{C}^{m-2}_{\mathcal{L}_2} & \mathcal{C}^{m-1}_{\mathcal{L}_2} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ \mathcal{C}^1_{\mathcal{L}_{m-1}} & \mathcal{C}^2_{\mathcal{L}_{m-1}} & \ldots & \mathcal{C}^{m-2}_{\mathcal{L}_{m-1}} & \mathcal{C}^{m-1}_{\mathcal{L}_{m-1}} \\ \mathcal{C}^1_{\mathcal{L}_m} & \mathcal{C}^2_{\mathcal{L}_m} & \ldots & \mathcal{C}^{m-2}_{\mathcal{L}_m} & \mathcal{C}^{m-1}_{\mathcal{L}_m} \end{pmatrix}$$

we can now reduce the number of unknowns by restraining the coordinates as follows:

$$\begin{pmatrix} 0 & 0 & \ldots & 0 & 0 \\ \mathcal{C}^1_{\mathcal{L}_2} & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ \mathcal{C}^1_{\mathcal{L}_{m-1}} & \mathcal{C}^2_{\mathcal{L}_{m-1}} & \ldots & \mathcal{C}^{m-2}_{\mathcal{L}_{m-1}} & 0 \\ \mathcal{C}^1_{\mathcal{L}_m} & \mathcal{C}^2_{\mathcal{L}_m} & \ldots & \mathcal{C}^{m-2}_{\mathcal{L}_m} & \mathcal{C}^{m-1}_{\mathcal{L}_m} \end{pmatrix}$$

By restraining the landmark coordinates we have eliminated rotation and translation. We have also reduced the number of unknowns by half, from $m(m-1)$ to $m(m-1)/2$.

This result alone leads to a noticeable acceleration of the convergence of GNP's function minimization. As we will show in the following section, next to the accelerated computation, there is also an explicit solution to (3.2) which can be computed directly.

3.4.2 Calculating the Simplex Coordinates

After eliminating rotation and translation, the equation system (3.2) can be simplified as follows:

$$
\begin{aligned}
\mathcal{C}^1_{\mathcal{L}_1} &= 0 \\
&\vdots \\
\mathcal{C}^{m-1}_{\mathcal{L}_1} &= 0 \\
(\mathcal{C}^1_{\mathcal{L}_2})^2 &= (d_{\mathcal{L}_1,\mathcal{L}_2})^2 \\
\mathcal{C}^2_{\mathcal{L}_2} &= 0 \\
&\vdots \\
\mathcal{C}^{m-1}_{\mathcal{L}_2} &= 0 \\
(\mathcal{C}^1_{\mathcal{L}_3})^2 + (\mathcal{C}^2_{\mathcal{L}_3})^2 &= (d_{\mathcal{L}_1,\mathcal{L}_3})^2 \\
&\vdots \\
\sum_{i=1}^{m-1} (\mathcal{C}^i_{\mathcal{L}_m})^2 &= (d_{\mathcal{L}_1,\mathcal{L}_m})^2 \\
(\mathcal{C}^1_{\mathcal{L}_2} - \mathcal{C}^1_{\mathcal{L}_3})^2 + (\mathcal{C}^2_{\mathcal{L}_3})^2 &= (d_{\mathcal{L}_2,\mathcal{L}_3})^2 \\
(\mathcal{C}^1_{\mathcal{L}_2} - \mathcal{C}^1_{\mathcal{L}_4})^2 + (\mathcal{C}^2_{\mathcal{L}_4})^2 + (\mathcal{C}^3_{\mathcal{L}_4})^2 &= (d_{\mathcal{L}_2,\mathcal{L}_4})^2 \\
\sum_{i=1}^{m-1} (c^i_{\mathcal{L}_2} - c^i_{\mathcal{L}_3})^2 &= (d_{\mathcal{L}_2,\mathcal{L}_3})^2 \\
&\vdots \\
\mathcal{C}^{m-1}_{\mathcal{L}_{m-1}} &= 0 \\
\sum_{i=1}^{m-1} (\mathcal{C}^i_{\mathcal{L}_{m-1}} - \mathcal{C}^i_{\mathcal{L}_m})^2 &= (d_{\mathcal{L}_{m-1},\mathcal{L}_m})^2
\end{aligned}
$$

As mentioned in 3.4.1, the position of the first landmark is set to the origin of $\mathcal{S}$. Since the second landmark is positioned on the first-dimension axis, the following equation delivers the first coordinate of the landmark:

$$\mathcal{C}^1_{\mathcal{L}_2} = \pm d_{\mathcal{L}_1,\mathcal{L}_2}$$

To determine the position of the third landmark, we have the following equations:

$$(\mathcal{C}^1_{\mathcal{L}_3})^2 + (\mathcal{C}^2_{\mathcal{L}_3})^2 = (d_{\mathcal{L}_1,\mathcal{L}_3})^2 \tag{3.3}$$

$$(\mathcal{C}^1_{\mathcal{L}_2} - \mathcal{C}^1_{\mathcal{L}_3})^2 + (\mathcal{C}^2_{\mathcal{L}_3})^2 = (d_{\mathcal{L}_2,\mathcal{L}_3})^2 \tag{3.4}$$

By subtracting (3.3) from (3.4) we obtain the first coordinate of $\mathcal{L}_3$:

$$\mathcal{C}^1_{\mathcal{L}_3} = -\frac{(d_{\mathcal{L}_2,\mathcal{L}_3})^2 - (d_{\mathcal{L}_1,\mathcal{L}_3})^2 - (\mathcal{C}^1_{\mathcal{L}_2})^2}{2\mathcal{C}^1_{\mathcal{L}_2}}$$

The second coordinate of $\mathcal{L}_3$ can be obtained by replacing the computed value of $c^1_{\mathcal{L}_3}$ in (3.3):

$$\mathcal{C}^2_{\mathcal{L}_3} = \pm\sqrt{(d_{\mathcal{L}_1,\mathcal{L}_3})^2 - (\mathcal{C}^1_{\mathcal{L}_3})^2}$$

In general, once we have determined the coordinates of the first k landmarks $(\mathcal{C}^{\mathcal{S}}_{\mathcal{L}_1}, \ldots, \mathcal{C}^{\mathcal{S}}_{\mathcal{L}_k})$, the position $\mathcal{C}^{\mathcal{S}}_{\mathcal{L}_{k+1}} := (\mathcal{C}^1_{\mathcal{L}_{k+1}}, \ldots, \mathcal{C}^k_{\mathcal{L}_{k+1}}, 0, \ldots, 0)$ of landmark $\mathcal{L}_{k+1}$ can be found by solving the following equation system:

$$\sum_{i=1}^{k}(\mathcal{C}^i_{\mathcal{L}_{k+1}})^2 = (d_{\mathcal{L}_1,\mathcal{L}_{k+1}})^2 \tag{3.5}$$

$$(\mathcal{C}^1_{\mathcal{L}_2} - \mathcal{C}^1_{\mathcal{L}_{k+1}})^2 + \sum_{i=2}^{m-1}(\mathcal{C}^i_{\mathcal{L}_{k+1}})^2 = (d_{\mathcal{L}_2,\mathcal{L}_{k+1}})^2 \tag{3.6}$$

$$\begin{array}{l}(\mathcal{C}^1_{\mathcal{L}_3} - \mathcal{C}^1_{\mathcal{L}_{k+1}})^2 + \\ +(\mathcal{C}^2_{\mathcal{L}_3} - \mathcal{C}^2_{\mathcal{L}_{k+1}})^2 + \\ +\sum_{i=3}^{k}(\mathcal{C}^i_{\mathcal{L}_{k+1}})^2\end{array} = (d_{\mathcal{L}_2,\mathcal{L}_{k+1}})^2 \tag{3.7}$$

$$\vdots$$

$$\sum_{i=1}^{k-2}(\mathcal{C}^i_{\mathcal{L}_k} - \mathcal{C}^i_{\mathcal{L}_k})^2 + (\mathcal{C}^{k-1}_{\mathcal{L}_{k+1}})^2 = (d_{\mathcal{L}_k,\mathcal{L}_{k+1}})^2$$

To calculate $\mathcal{C}^1_{\mathcal{L}_k}$, we subtract (3.5) from (3.6) The result is:

$$\mathcal{C}^1_{\mathcal{L}_{k+1}} = \frac{(d_{\mathcal{L}_1,\mathcal{L}_{k+1}})^2 - (d_{\mathcal{L}_2,\mathcal{L}_{k+1}})^2 + (\mathcal{C}^1_{\mathcal{L}_2})^2}{2\mathcal{C}^1_{\mathcal{L}_2}}$$

To calculate $\mathcal{C}^2_{\mathcal{L}_k}$, we subtract (3.5) from (3.7), which leads to:

$$\begin{array}{rl}\mathcal{C}^2_{\mathcal{L}_{k+1}} = & (2\mathcal{C}^2_{\mathcal{L}_3})^{-1}\big[(d_{\mathcal{L}_1,\mathcal{L}_{k+1}})^2 - (d_{\mathcal{L}_3,\mathcal{L}_{k+1}})^2 - \\ & -2\mathcal{C}^1_{\mathcal{L}_3}\mathcal{C}^1_{\mathcal{L}_{k+1}} + (\mathcal{C}^1_{\mathcal{L}_3})^2 + (\mathcal{C}^2_{\mathcal{L}_3})^2\big]\end{array} \tag{3.8}$$

This procedure can be used for all coordinates $\mathcal{C}^1_{\mathcal{L}_{k+1}}, \ldots, \mathcal{C}^{k-1}_{\mathcal{L}_{k+1}}$. For the last coordinate $\mathcal{C}^k_{\mathcal{L}_{k+1}}$, we use the computed values and replace them in (3.5):

$$\mathcal{C}^k_{\mathcal{L}_{k+1}} = \pm\sqrt{(d_{\mathcal{L}_1,\mathcal{L}_{k+1}})^2 - \sum_{i=1}^{k-1}(\mathcal{C}^i_{\mathcal{L}_{k+1}})^2}$$

The generalized formula for calculating the simplex coordinates derived from this procedure is:

$$\mathcal{C}^j_{\mathcal{L}_i} = \begin{cases} 0 & , j \geq i \\ (2\mathcal{C}^j_{\mathcal{L}_{j+1}})^{-1}\big[(d_{\mathcal{L}_1,\mathcal{L}_i})^2 - \\ -(d_{\mathcal{L}_{j+1},\mathcal{L}_i})^2 - \\ -\sum_{k=1}^{j-1} 2(\mathcal{C}^k_{\mathcal{L}_{j+1}})(\mathcal{C}^k_{\mathcal{L}_i}) + & , j < i-1 \\ +\sum_{k=1}^{j} (\mathcal{C}^k_{\mathcal{L}_{j+1}})^2\big] \\ \pm\sqrt{(d_{\mathcal{L}_1,\mathcal{L}_i})^2 - \sum_{k=1}^{j-1} (\mathcal{C}^k_{\mathcal{L}_i})^2} & , j = i-1 \end{cases} \tag{3.9}$$

With $\frac{m\cdot(m-1)}{2}$ computations of $\mathcal{C}^j_{\mathcal{L}_i}$, this formula iteratively yields the coordinates of the landmarks for the given distances.

3.5 Distances and Metric Spaces

Distances derived from real-world measurements, such as distances to GPS satellites, do not usually yield exact positioning. This is due to measurement errors. Still, we know that the measurements originated in a three-dimensional environment in which the triangle inequality holds. The problem with coordinates-based RTT prediction is that we are trying to fit "distance" measurements into a k-dimensional Euclidean space without knowing the dimensionality of the space in advance, i.e. without knowing the value of k.

In GNP, the authors base the "optimal" number of dimensions as well as landmarks that can be used on the results of their experiments. Here, we will show how to infer the optimal number of dimensions for embedding landmarks using only a distance matrix.

3.5.1 Handling Triangle Inequality Violations

As mentioned earlier, it is not possible to reconstruct an $(m-1)$-dimensional simplex for just any arbitrary distance matrix representing the distances between m points. The reason for this is that projecting a distance matrix into an Euclidean space (equivalent to constructing a simplex from a distance matrix) poses strict requirements on the distances. We described those requirements in chapter 1.5.1.

To still be able to have a set of landmarks, one must reduce the number of dimensions. For example, it is obvious that the triangle inequality does not hold for the following dis-

tance matrix ($5 > 3 + 1$):

$$\begin{pmatrix} 0 & 1 & 3 \\ 1 & 0 & 5 \\ 3 & 5 & 0 \end{pmatrix}$$

To still be able to have landmarks we can try to reduce the number of dimensions. Instead of constructing a 2-simplex (a triangle - which is impossible since the triangle inequality does not hold) we could construct a 1-simplex (a line segment), which serves as a "base" for positioning the third landmark. We could then position the third point on the straight line such that the embedding-error (e.g. the square error between given and calculated distances) is minimized. For example, we could assign the following coordinates to the landmarks:

$$\begin{pmatrix} c_{\mathcal{L}_1} \\ c_{\mathcal{L}_2} \\ c_{\mathcal{L}_3} \end{pmatrix} = \begin{pmatrix} (0) \\ (1) \\ (3.5) \end{pmatrix}$$

This solution is not the only one. There are actually three possible solutions – depending on which pair of landmarks is chosen as the "base". It is also important to note that the possible solutions are not of equal value when it comes to the total square error of such an embedding. Since we are interested in minimizing the total error, the optimal strategy is to use the base for which the total error is the smallest – in this case it is the 1-simplex which is formed by landmarks $\mathcal{L}_1$ and $\mathcal{L}_3$. Whatever solution is chosen, it is usually not an optimal one, but is a very good starting point for further minimization – for example using the Downhill Simplex method.

3.5.2 Handling Simplex Inequality Violations

Unfortunately, the triangle inequality is not the only constraint with constructing a simplex. For example, for the following distance matrix:

$$\begin{pmatrix} 0 & 3 & 3 & 1.6 \\ 3 & 0 & 3 & 1.6 \\ 3 & 3 & 0 & 1.6 \\ 1.6 & 1.6 & 1.6 & 0 \end{pmatrix}$$

the triangle inequality holds for every triple of distances between the landmarks ($3 \leq 3 + 3, \quad 3 \leq 3 + 1.6, \quad 1.6 \leq 3 + 1.6, \quad 1.6 \leq 3 + 3$). Still we are not able to construct a 3-simplex (tetrahedron) with the given distances, because the triangle inequality is the necessary but not sufficient condition for constructing a k-simplex ($k \geq 2$). The sufficient condition in three dimensions would be that the area of every tetrahedron side (triangle) must be smaller than or equal to the sum of the areas of all other sides [46]. This last inequality does not hold for our distance matrix.

A more generalized recursive inequality can be defined for any k-simplex: The hypervolume of every "side" (a $(k-1)$-simplex) of a simplex must be smaller or equal to the sum of the hypervolumes of all other simplex-"sides". This inequality is also known as the

simplex inequality [47]. The simplex inequality is actually recursive, since, to be able to compute the hyper-volume of a simplex, one must be able to construct a simplex. The recursion ends with the triangle inequality for a 2-simplex. By knowing this, we can construct the landmark coordinates by reducing the dimensionality as described in Section 3.5.1. In this case, the "base" is a 2-simplex (a triangle) and the fourth landmark is positioned in the plane defined by the positions of the "base" points.

3.5.3 The Cayley-Menger Determinant

Now that we know the sufficient condition for constructing a simplex, we should be able to find a set of landmarks that can be used as a base. Unfortunately, there is one small detail of the simplex inequality which is still not trivial: How do we calculate the hyper-volume of a $(k-1)$-simplex given only the lengths of the edges of the simplex?

In the case of a 1-simplex or a 2-simplex the solution is very simple. The hyper-volume of a line segment is its length; the hyper-volume of a triangle is its area, which can be calculated using Heron's formula. For a tetrahedron (a 3-simplex), there is Tartaglia's formula. However, we still need a general approach which delivers the volume of a k-simplex. Cayley-Menger Determinant [46] is such an approach.

The value of Cayley-Menger determinant is the square of the hyper-volume of a k-dimensional simplex S_k. $V^2(S_k)$ is based only on the lengths of the edges of the simplex. The square hyper-volume of S_k is defined as

$$V^2(S_k) = \frac{(-1)^{k+1}}{2k(k!)^2} \det(B) \tag{3.10}$$

B is a matrix obtained by bordering a quadratic distance matrix with a top row $(0, 1, \ldots, 1)$ and a left column $(0, 1, \ldots, 1)^T$. For example for a 3-simplex the B matrix would look like:

$$B = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & (d_{\mathcal{L}_1,\mathcal{L}_2})^2 & (d_{\mathcal{L}_1,\mathcal{L}_3})^2 & (d_{\mathcal{L}_1,\mathcal{L}_4})^2 \\ 1 & (d_{\mathcal{L}_2,\mathcal{L}_1})^2 & 0 & (d_{\mathcal{L}_2,\mathcal{L}_3})^2 & (d_{\mathcal{L}_2,\mathcal{L}_4})^2 \\ 1 & (d_{\mathcal{L}_3,\mathcal{L}_1})^2 & (d_{\mathcal{L}_3,\mathcal{L}_2})^2 & 0 & (d_{\mathcal{L}_3,\mathcal{L}_4})^2 \\ 1 & (d_{\mathcal{L}_4,\mathcal{L}_1})^2 & (d_{\mathcal{L}_4,\mathcal{L}_2})^2 & (d_{\mathcal{L}_4,\mathcal{L}_3})^2 & 0 \end{pmatrix}$$

The Cayley-Menger determinant is very convenient because of the way the hyper-volume of a simplex is defined. The volume of a simplex is proportional to the volume of a k-dimensional parallelepiped with the same edge lengths. The hyper-volume of a k-dimensional parallelepiped is defined as the parallelepiped's height in one dimension multiplied with the hyper-volume of the parallelepiped's base. The square of a hyper-volume contains not only the information about the simplex hyper-volume, but also the information about the "heights" of the simplex in lower dimensions.

This information is necessary to be able to draw conclusions about the co-planarity (in the sense of hyper-planes) of a simplex. It can be used to determine the optimal number of dimensions which should be utilized to construct the simplex. For example, if we calculate

the square hyper-volume for the distances of the example from Section 3.5.2, we would obtain a negative value.

The negative value of the square hyper-volume means that the volume of the simplex can only be represented as a complex number. It also tells us that at least one "height" of the simplex must be negative, meaning that we will not be able to construct an optimal simplex. A square volume of value zero tells us that at least one height of the simplex is zero, which means that the points are coplanar (in the sense of hyper-planes) and can be represented with at least one dimension less than k. A positive square hyper-volume indicates that it *may* be possible to construct an ideal simplex – it does not indicate that it definitely *is* possible. There may be an even number of negative heights in the sub-simplexes which would still result in a positive square hyper-volume since the product of an even number of negative numbers is positive. For example, the Cayley-Menger determinant for the following distance matrix computes a positive square hyper-volume, although the triangle equation does not hold $(77 > 50 + 25)$:

$$\begin{pmatrix} 0 & 50 & 77 & 36 \\ 50 & 0 & 25 & 5 \\ 77 & 25 & 0 & 35 \\ 36 & 5 & 35 & 0 \end{pmatrix}$$

This is why we state that the positive square hyper-volume only indicates the possibility to construct a hyper-volume. To be sure one has to recursively check if all hyper-volumes of all sub-simplexes (simplexes in lower dimensions that can be built using a subset of the simplex points) are positive.

3.6 Fast Landmark Positioning

The results presented in Sections 3.4 and 3.5 allow us to define two algorithms. Algorithm 3.1 determines an ideal value for k, where k is the dimension of the Euclidean space into which the landmarks are embedded. It does this by finding all maximal subsets of landmarks which allow the construction of an ideal simplex.

Knowing the optimal value of k helps us further accelerate the process of finding landmark positions because it reduces the number of variables in the function minimization. Furthermore, we can now construct a base simplex and estimate the coordinates of those landmarks which are not a part of the base simplex. This helps us specify a good starting point for the function minimization and further optimizes the process. Thus, Algorithm 3.2 uses the "best" simplex found by Algorithm 3.1 as the base for embedding and to find a good starting point for the function minimization of GNP. The motivation behind the two algorithms is to reduce the total overhead of the system by finding an optimal number of dimensions for representing the distances between the landmarks and by providing a good starting point for the function minimization.

Algorithm 3.1: *FindMaxSimplexes*: Algorithm for finding all simplexes of the maximum dimensionality

Input : Distance matrix D containing all distances between m landmarks
Output: A set of simplexes S that contains all simplexes (each of them represented as a set of indexes) that can be constructed from D

$S \leftarrow \{\{x\} | x \in \{1, \cdots, m\}\}$;
for $i \leftarrow 1$ **to** m **do**
  for $x \in S$ **do**
    /* Add a new simplex if its square volume is positive */
    if SimplexQuadVolume$(D, x \cup \{i\}) > 0$ **then**
      $S \leftarrow S \cup \{x \cup \{i\}\}$;
    end
  end
  /* Find the maximal simplex size */
  max $\leftarrow \max(\|x\| : x \in S)$;
  /* Remove all simplexes that cannot grow larger than the maxi simplex */
  $S \leftarrow S \backslash \{x : x \in S, \|x\| + (m - i) <$ max$\}$;
end
return S;

3.7 Evaluation

3.7.1 Evaluation Scenarios

In this Section, we present how our fast landmark positioning algorithm compares experimentally to the landmark positioning algorithm used by GNP. Our focus is on the accuracy of the landmark positioning (the value of the error function) and the computational overhead (CPU time used to compute the landmark positions). We base our comparison on data obtained from the PlanetLab "All-Sites-Pings" experiment [41].

To obtain comparable results, we implemented the GNP algorithm and both of our algorithms in the programming language Perl. For the function minimization we used the Perl module Math::Amoeba obtained from CPAN [48], which implements the Downhill Simplex algorithm. For each function minimization we used following parameters: $f_{\text{tol}} = 10^{-7}$ and a maximum eval count of 40000 (for details see documentation of Math::Amoeba). All execution time measurements were performed on a computer with a 3 GHz Pentium 4 CPU running Linux.

The methodology for comparison is always the same: We vary the number of landmarks m between 4 and 20. For each m, we randomly choose 20 different subsets of end systems out of the total 125 end systems available in our data-set. Each of these subsets contains m end systems which we now use as landmarks (landmark-set). For each such landmark-

Algorithm 3.2: *FindLandmarkPositions*: Algorithm for fast landmark positioning

Input : Distance matrix D containing all distances between m landmarks
Output: A good starting point P for function minimization and the optimal number of dimensions

find all simplexes with the maximum dimensionality using the `FindMaxSimplexes` algorithm;

for *each found simplex* **do**

determine the coordinates of the landmarks that are in the simplex using (3.9);

determine the coordinates of the remaining landmarks using GNP host positioning, where the hosts for the simplex are used as the landmarks;

add the positions of the landmarks to the set of the potential start-points;

end

choose the best start-point amount the potential starting points by finding the one with the smallest total square error of the embedding;

set, we calculate the coordinates of the landmarks and use the UNIX time utility to record the time needed to execute the process. In our simulations, we used each of the following landmark positioning algorithms to calculate the landmark coordinates:

- *MAX*: The original GNP landmark positioning where m landmarks are positioned in an $(m-1)$-dimensional space. The starting point for the function minimization is chosen randomly. The result of this landmark positioning should have the smallest total positioning error, since we have the maximum number of dimensions available. This positioning is also the slowest one, since we do not have a reasonable starting point for the function minimization and the number of variables is the maximum $(m(m-1))$.

- *OPTDIM*: This is the GNP landmark positioning where m landmarks are positioned in a k-dimensional space. The optimal number of dimensions k is determined using Algorithm 3.1. The starting point for the function minimization is chosen randomly. This landmark positioning should be much faster than *MAX* since the optimal number of dimensions is usually much lower than the maximal number of dimensions. The result of this landmark positioning should have the same total positioning error as *MAX* if our algorithm for determining an optimal number of landmarks is correct. The execution time for this algorithm contains both the execution time for finding optimal number of dimensions and the function minimization itself.

- *OPTDIM-SP*: This is the same landmark positioning as *OPTDIM*, however, its starting point for the function minimization is determined by Algorithm 3.2. The execution time for calculating the starting point is also a part of the total execution time. If our assumption is correct, this algorithm should be faster than *OPTDIM*, because the

function minimization should be faster when a good starting point is provided.

- *OPTDIM-SP-R*: This landmark positioning algorithm does the same function minimization as *OPTDIM-SP* but eliminates the rotation and translation of the resulting positions (for details see Section 3.4.1). Our assumption was that this algorithm would be the fastest, since it applies every optimization we proposed in this chapter: optimal number of dimensions, starting point for the function minimization and reducing the number of variables used in the function minimization.

Since there are two flavours of GNP (for details see Section 2.1.2), we performed our tests using both error functions for GNP. We refer to GNP using the standard distance square error function f_e as plain GNP. Normalized GNP is the GNP which uses the square error of normalized distances.

3.7.2 Correctness

The first thing that interests us is whether our algorithm really finds an optimal number of dimensions. To verify this, we compare, for each number of landmarks m, the total error of the landmark coordinates computed by *OPTDIM-SP* with the total error that is obtained by performing GNP using $m-1$ dimensions (*MAX*). If our algorithm is correct, the total error of our landmark positioning *OPTDIM-SP* should be equal to or better than the total error of *MAX*.

Since the expected value of the total square error varies considerably, depending on the choice of landmarks, we performed Wilcoxon signed-rank [49] statistical tests for each number of landmarks. The reason for choosing the Wilcoxon signed-rank test is that we have paired measurements (total error of landmark positioning for the same set of landmarks for each algorithm) and that the test makes no assumption about the distribution of the data.

With the test we are trying to detect whether the medians of the total errors of *MAX* and *OPTDIM-SP* are different. To be able to detect all kinds of differences we performed three tests for each pair of value sets.

- The first test ($t_<$) tests whether the median of *MAX* is lower than the median of *OPTDIM-SP*.

  In this case, our conservative hypothesis is $H_0^< : \mu(\textit{MAX}) \geq \mu(\textit{OPTDIM-SP})$. The alternative hypothesis is $H_1^< : \mu(\textit{MAX}) < \mu(\textit{OPTDIM-SP})$.

- The second test ($t_>$) determines whether the median of *MAX* is higher than the median of *OPTDIM-SP*.

  The hypotheses of this thest are: $H_0^> : \mu(\textit{MAX}) \leq \mu(\textit{OPTDIM-SP})$ and $H_1^> : \mu(\textit{MAX}) > \mu(\textit{OPTDIM-SP})$.

- The third test ($t_\neq$) checks whether the medians of *MAX* and *OPTDIM-SP* differ significantly.

In this test, we have the hypotheses: $H_0^{\neq}$: $\mu(MAX) = \mu(OPTDIM\text{-}SP)$ and $H_1^{\neq}$: $\mu(MAX) \neq \mu(OPTDIM\text{-}SP)$. To determine which hypothesis is right we have to compare the p values with the significance level ($\alpha = 0.05$).

For example, we obtain the following p values for 5 landmarks and plain GNP: 0.978 for $t_>$, 0.022 for $t_<$ and 0.044 for $t_{\neq}$. From this, we can deduct that $\mu(MAX)$ is significantly higher than $\mu(OPTDIM\text{-}SP)$, because the p values for the tests $t_>$ and $t_{\neq}$ are below the significance threshold of 0.05. This means that the hypotheses $H_1^{>}$ and $H_1^{\neq}$ are correct.

We performed these statistical tests for each number of landmarks and for both flavors of GNP (plain and normalized). We have computed the p values of the tests using the *wilcoxon_test()* statistical function of octave [50], a mathematical software. All p values of the tests are presented in Table 3.1. As can be seen, not only does our algorithm for finding

Table 3.1: Wilcoxon Matched Pairs Signed-Rank test p values of the total error for MAX and OPTDIM landmark positioning.

	MAX vs OPTDIM-SP (Plain GNP)			MAX vs.OPTDIM-SP (Normalized GNP)		
	$p(t_<)$	$p(t_>)$	$p(t_{\neq})$	$p(t_<)$	$p(t_>)$	$p(t_{\neq})$
4	0.617	0.383	0.765	0.206	0.794	0.411
5	0.978	0.022	0.044	0.999	0.001	0.002
6	0.997	0.003	0.005	0.999	0.001	0.001
7	1.000	0.000	0.001	0.997	0.003	0.006
8	1.000	0.000	0.001	1.000	0.000	0.000
9	1.000	0.000	0.001	0.996	0.004	0.009
10	1.000	0.000	0.000	1.000	0.000	0.000
11	0.999	0.001	0.002	1.000	0.000	0.000
12	1.000	0.000	0.000	0.999	0.001	0.001
13	0.999	0.001	0.001	0.999	0.001	0.001
14	1.000	0.000	0.001	1.000	0.000	0.000
15	1.000	0.000	0.000	1.000	0.000	0.000
16	1.000	0.000	0.000	1.000	0.000	0.000
17	1.000	0.000	0.000	1.000	0.000	0.000
18	1.000	0.000	0.000	1.000	0.000	0.000
19	1.000	0.000	0.000	1.000	0.000	0.000
20	1.000	0.000	0.000	1.000	0.000	0.000

an optimal number of dimensions seem to be correct, but also has a significantly lower total error than the GNP which uses the maximal number of dimensions for each number of landmarks.

We performed the same tests to compare the performance of *OPTDIM-SP-R* with *OPTDIM-SP*. To our surprise, the tests showed that *OPTDIM-SP-R* results in significantly larger errors than *OPTDIM-SP*. Further investigations have shown that the error difference is very small for each sample. The median of the RTT prediction error between *OPTDIM-SP* and *OPTDIM-SP-R* was less than 1% of the predicted distance. The value distribution of the

prediction error was very asymmetric – most of the values were very low (less than 1.5%) and we only had a few outliers with very high values.

We assumed that there are two reasons for this result. The first reason is the decreased numerical precision of the Downhill Simplex function minimization when working with a reduced number of variables. By reducing the number variables, we have limited Downhill Simplex in its freedom of optimization. In other words, in the original GNP, there exists an infinite number of solutions. The algorithm starts converging towards one of them. While optimizing in other dimensions, it may then choose to "skip" that solution and converge towards another one. By reducing the number of variables, we eliminated that option from the algorithm. Thus, most of the small error values could be explained.

The reason for the outliers with high errors is the increased probability that the Downhill Simplex becomes stuck in a local minimum when the number of possible solutions is reduced. Depending on the starting point for the function minimization (initial simplex), the Downhill Simplex method may converge to a local minimum. There is no guarantee that this local minimum is also the global one. When the number of possible solutions is not reduced, there is a higher probability that the Downhill Simplex will find and converge toward a better solution in the vicinity of the non-optimal solution. This shows that reducing the number of the possible solutions does not always have positive effects.

3.7.3 Processing Performance

Our accuracy analysis indicates that our error prediction is not worse than the best case of the GNP landmark positioning. Now we are interested in whether our landmark positioning has any performance gains compared to the standard GNP positioning. To verify this, we compared the execution time (in terms of CPU time spent by the execution process) for the following positioning implementations: *MAX*, *OPTDIM* and *OPTDIM-SP*.

If our assumptions were correct, the execution time of *MAX* should be the worst of all alternatives, followed by *OPTDIM* and *OPTDIM-SP* should be the fastest algorithm.

We present the results of the experiments as plots: the execution times are shown in Figures 3.5 and 3.6, the number of dimensions in Figure 3.7. The plots summarize the results of 20 experiments grouped for each number of landmarks using bars. The horizontal line at the top of the bar represents the maximum of all values, the line in the middle represents the median, and the line at the bottom of the bar indicates the minimum of all values. The medians of the bars for the different experiments are connected with a line.

For the sake of completeness, in Figure 3.7, we also present the optimal number of dimensions, as they were calculated by our algorithm for the experiments.

3.8 Conclusion

In this chapter, we showed that it is possible to determine the optimal number of dimensions k for embedding landmarks in an Euclidean space. We also showed that it is possible to find a good starting point for the function minimization. Based on this, we have defined two algorithms. The first algorithm finds an optimal number of dimensions k for embedding

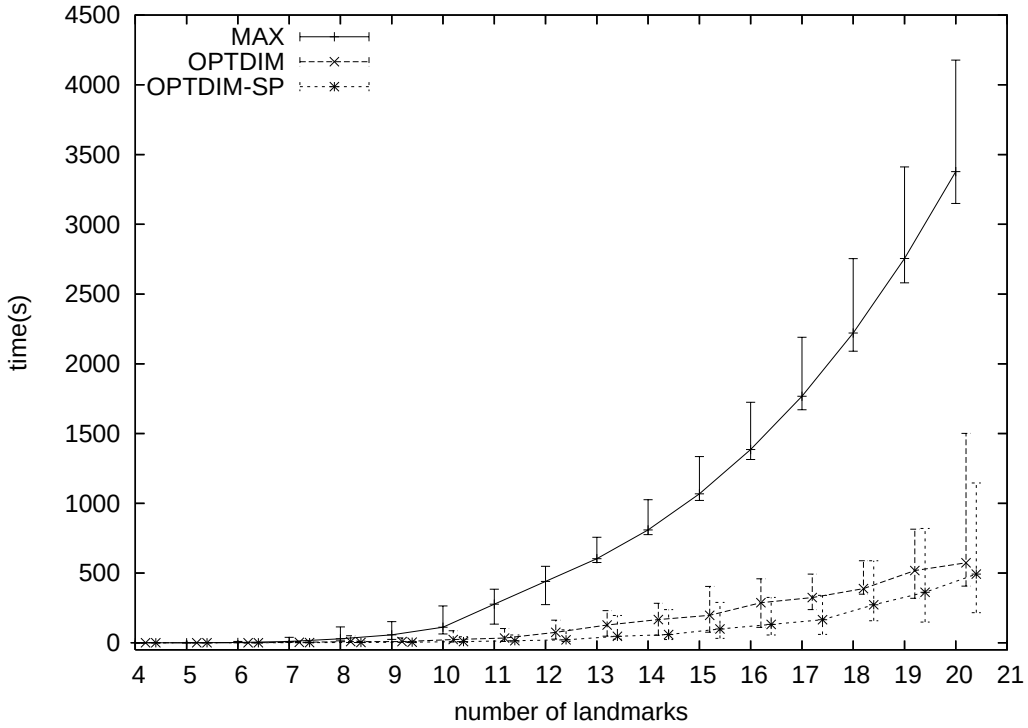

Figure 3.5: Performance comparison for the plain GNP algorithm

landmarks in a Euclidean space and all sets of landmarks that can be used as a base for that embedding. The second algorithm finds a good starting point for the function minimization used in GNP to find the landmark positions. We have validated our algorithms by using data obtained from PlanetLab "All-Sites-Ping" experiment. The results of our evaluation show that both algorithms proposed by us are correct and able to accelerate the computation of the positions of landmarks in GNP. Our results also show that reducing the number of possible solutions for the function minimization can introduce an additional embedding-error to the landmark positioning when Downhill Simplex is used for the function minimization.

As mentioned above, in our experiments, we always assumed that the objective function used by GNP may have more than one local minimum. In the next chapter, we will show that objective functions in GNP do indeed have multiple local minima. We will propose an algorithm which is designed to find most of those local minima and, consequently, has a high chance of also finding the global minimum of the objective function.

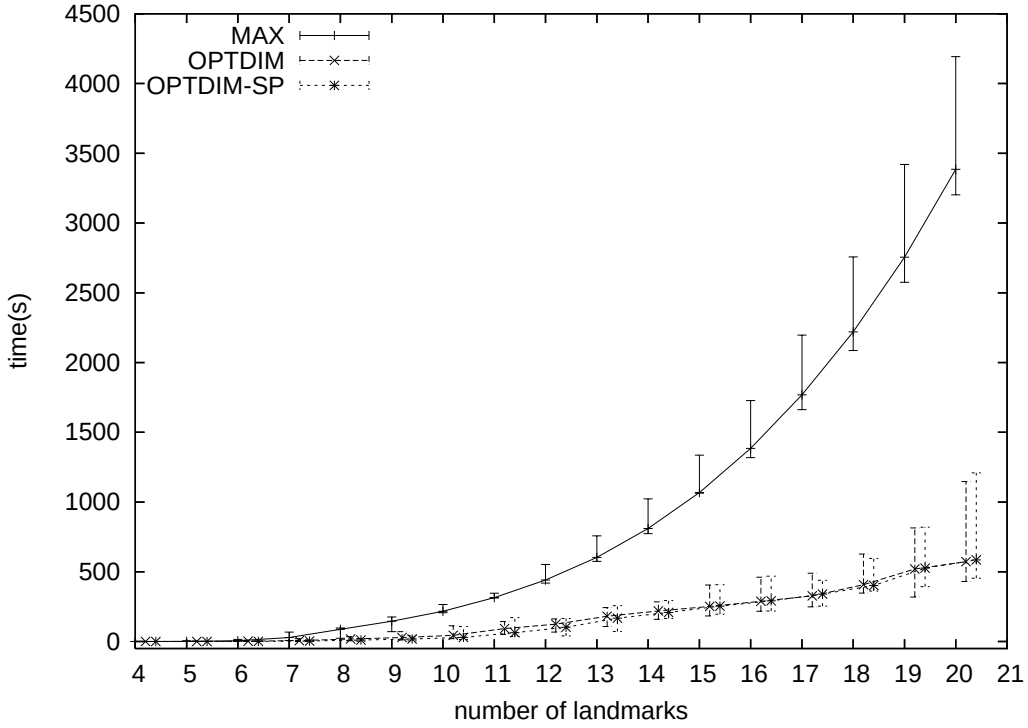

Figure 3.6: Performance comparison for the normalized GNP algorithm

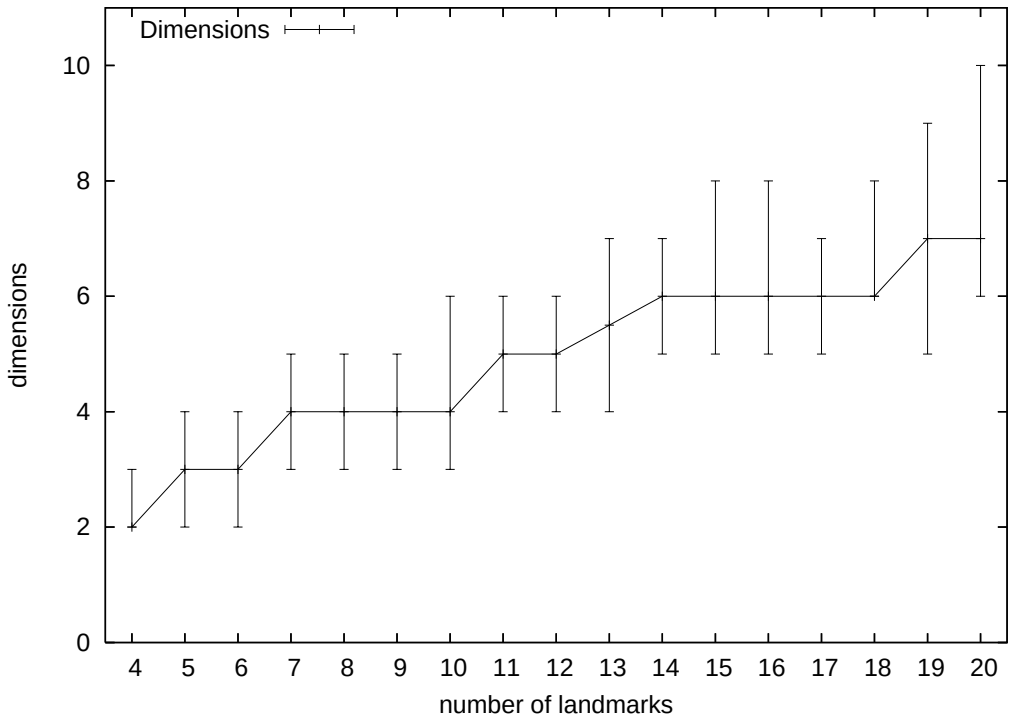

Figure 3.7: Dimensions for number of landmarks

Chapter 4

Enhancing Round-Trip Time Prediction By Using Global Function Minimization

4.1 Introduction

In the last chapter, we tackled the problem of computational overhead when positioning landmarks in GNP. In this chapter, we identify another problem of GNP: The existence of multiple local minima in the objective function, i.e. the function which is minimized to determine the positions of the end systems in the virtual space.

As mentioned in Section 2.1.2, GNP uses function minimization to identify the positions of landmarks (denoted by $\mathcal{L}_1, \ldots, \mathcal{L}_m$) and other end systems (denoted here by $\mathcal{H}$) in a k-dimensional Euclidean space. The positions of the landmarks are calculated from the full RTT distance matrix by minimizing the square error function (2.2). All other end systems derive all their RTT distances from measurements to the landmarks (denoted by $d_{\mathcal{H},\mathcal{L}_1}, \ldots, d_{\mathcal{H},\mathcal{L}_m}$) and use this information to determine their position in the virtual space. The position of an end system in a virtual space (denoted by $\mathcal{C}_\mathcal{H} := (\mathcal{C}^1_\mathcal{H}, \ldots, \mathcal{C}^n_\mathcal{H})$) is determined by minimizing the error function (2.1).

For both landmarks and end system error functions, the functions are minimized using a numerical method. Here, the basic procedure is to start at one point in the parameter space of the function (known as a starting point) and find a better solution (a point where the function value is lower) by exploiting information inherent to the function, for example its first and second derivative, or an examination of the vicinity of the starting point (used in Downhill Simplex). If a better solution is found, the procedure is reiterated using the newly found position as the new starting point. This procedure is reiterated until either no better solution can be found or a specific criterion is fulfilled, e.g. the size of the last improvement drops below a predefined threshold, or until a maximum number of iterations has been performed.

The problem of numerical function minimizations is that they find a local minimum. If the function to be minimized has more than one minimum, this procedure finds exactly one minimum without any guarantee that this is also the global minimum.

By numerical means, it is very difficult to find the global minimum of an arbitrary

function. To identify the global minimum, we would have to enumerate all local minima and evaluate the function at all those points to discover the one with the minimal function value. And still the problem would remain: Even if there were an algorithm which was able to enumerate all local minima of any function; there are functions with an infinite number of local minima. Using this algorithm on such a function would result in a never-ending operation. To avoid this, almost all general numerical approaches for finding local minima terminate their calculation after a predetermined amount of time.

In this chapter, we outline the issues arising from the fact that the function used by GNP to determine the position of an end system in a virtual space contains several local minima. We also try to quantify the impact of using local instead of global minima on RTT predictions and evaluate the probability that a numerical method finds a non-global minimum. As a way of solving this last problem, we propose an algorithm which is tailor-made for the objective function. To find the global minimum it enumerates all or most of the local minima. Finally, to prove the effectiveness of our algorithm, we evaluate data collected in the Planet-Lab experiments.

4.2 Motivation

The mere existence of several local minima does not necessarily mean that RTT predictions generated by GNP are significantly different from RTT results which take local minima into account. We performed a series of experiments to evaluate the magnitude of the problem, i.e., to determine how many of the end systems are affected when choosing a local minimum instead of the global minimum. We were also interested in the question whether the quality of the RTT prediction degrades significantly if a local instead of the global minimum is used to embed the end systems.

4.2.1 How Frequent are Multiple Minima?

To determine the frequency of local minima in an objective function, we performed a so-called "monte carlo" minimum search. The idea is to perform a large number of numeric minimum searches – each time using a different, randomly chosen start point. If we performed an infinite number of such function minimizations, we could find every local minimum of any given function. Since we know that the number of minima is quite low (usually not higher than 5), performing a relatively small number of such minimizations should result in a good estimate about how many minima there are and where they are located. In our case, we used 400 function minimizations.

Because we use numerical methods, we still need a non-strict method to determine whether two minima are the same. In this experiment, we call two local minima m_1, m_2 equal if the following inequality holds: $d(\mathcal{C}_{m_1}, \mathcal{C}_{m_2}) < 2$ ms. We use the value of 2 since the units used in the all-pings experiment are milliseconds and a positioning error of two milliseconds is an acceptable compromise between the number of false positives and the prediction error – we are looking for errors of much larger magnitude. In other words, our method cannot distinguish between two local minima if the Euclidean distance between

them is less than two milliseconds. We do not want to know how many minima there are exactly. All we are interested in is the number of local minima which are located "far" away from each other.

In the beginning we used the Downhill Simplex method to minimize the function. As a result, we found a large amount of local minima located near each other (but more than 2 ms apart). Further investigation revealed that the main reason for this effect was not the existence of numerous local minima, but which method was used to minimize the function. It was often the Downhill Simplex method which became stuck at a point that was not a local minimum. This was most probably due to the collapse of the simplex. This is a known issue of the Downhill Simplex method. The used simplex "loses" at least one dimension because a "better" point is chosen, which is close to the plane defined by other points of the simplex. To eliminate this effect, we decided to use a more robust method to position the end systems: The Conjugate Gradient method yielded vastly improved results.

In our experiment, we took a snapshot of the RTT information measured by the PlanetLab all-pings experiment [41]. Since the data of the all-pings experiment is not complete, we removed some end systems from the data set in order to obtain a complete distance matrix which contains measured RTTs for each pair of end systems. The final distance matrix contained the complete distance information for 218 end systems scattered throughout the Internet. We used a fixed number of landmarks (12) and calculated their coordinates for varying dimensions (2 to 9) with the method proposed by GNP (Downhill Simplex). For all other end systems, which were not landmarks, we calculated the positions in the virtual space by minimizing the objective function (2.1).

This method allowed us to determine the percentage of end systems for which the objective function has more than one local minimum. Those results are presented in Figure 4.1. With the gathered data we were also able to estimate the probability of finding the global minimum when starting the minimization at some random point within the boundaries: We divided the number of random starting points that resulted in the global minimum by the total number of starting points (400). Figure 4.2 presents the average of those probabilities.

Our experiment showed that the number of the end systems for which the objective function has more than one minimum decreases with an increasing number of dimensions. The probability of finding the global minimum when starting at a random point within the boundaries is a more or less constant 60% for a number of dimensions between two and five. We can also see that for more than five dimensions the number of local minima found is practically zero. Unfortunately, since the computational complexity of a function minimization increases rapidly with the number of dimensions, we cannot simply increase the number of dimensions to reduce the probability of the existence of local minima.

4.2.2 Local Minima of the Objective Function

It may be tempting to believe that local minima exist solely due to violations of the triangle inequality of the measured data, which, in turn, is usually caused by policy-based routing in the Internet. To show that this is not the case, we have constructed the following example:

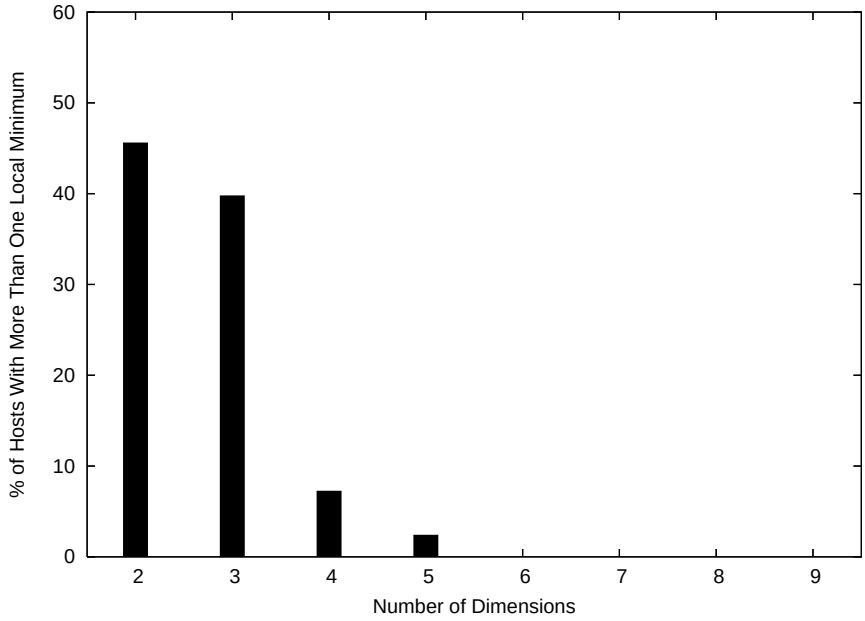

Figure 4.1: Percentage of End Systems With More Than One Local Minimum of the Objective Function.

We take a two-dimensional Euclidean space and four landmarks $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3, \mathcal{L}_4$ with the following coordinates: $\mathcal{C}_{\mathcal{L}_1}$: $(0, 180)$, $\mathcal{C}_{\mathcal{L}_2}$: $(0, -180)$, $\mathcal{C}_{\mathcal{L}_3}$: $(30, -30)$, $\mathcal{C}_{\mathcal{L}_4}$: $(30, 30)$ and an end systems with the coordinates $\mathcal{C}_{\mathcal{H}}$: $(-100, 0)$.

When we calculate the Euclidean distances between the end system and the four landmarks and use them (together with the landmark positions) to construct the usual objective function (2.1) used to embed end systems, we obtain a function which has two local minima (see Figure 4.3). The Figure clearly shows that the objective function has not only one, but two local minima. One of them is located at the real position of the end system $\mathcal{H}$ at $(-100, 0)$. The other is located behind the "pass" formed between the "peaks" created by the two nearest landmarks ($\mathcal{L}_3$ and $\mathcal{L}_4$). The function values at the two local minima are different. While the value of the objective function at the global minimum (at the original position of the end system $\mathcal{H}$) is zero, the value at the other local minimum is larger than zero, but surrounded by points with larger values.

At present, there are only very few ideas for generalized approaches to finding a global minimum [51]. None of these approaches uses the information about the morphology of the function. We think that in our particular case, i.e. where we have information about the morphology of the objective function, a better approach to finding the global minimum is to use the function's properties.

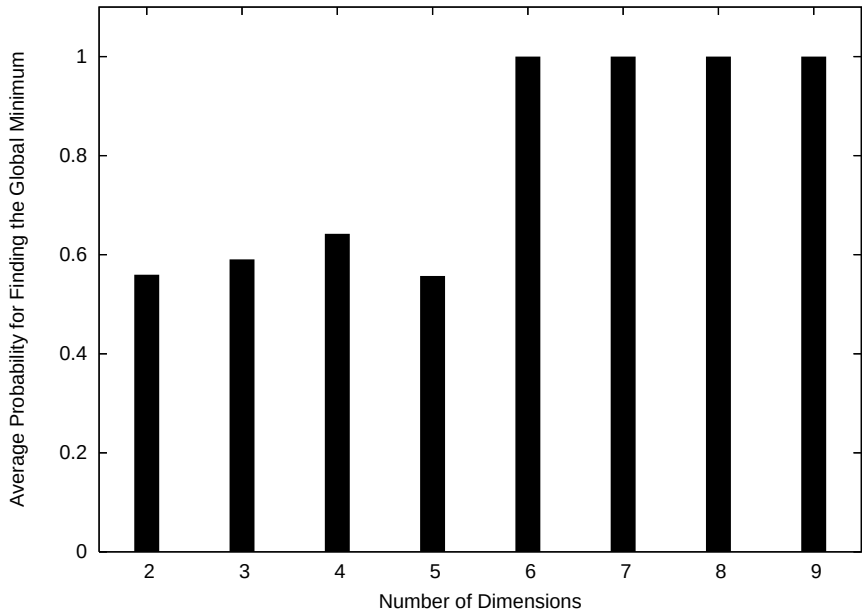

Figure 4.2: Average Probability for Finding the Global Minimum by Starting the Function Minimization at a Random Point

4.2.3 Impact of Local Minima on Round-Trip Time Prediction

As an illustration how the existence of local minima affects the accuracy of RTT predictions we show the cumulative distribution function (CDF) of the relative error of an RTT prediction for an end system with three local minima in two dimensions. Figure 4.4 shows the CDF of the relative error $\left|\frac{d(\mathcal{C}_{\mathcal{H}_1},\mathcal{C}_{\mathcal{H}_i})-d_{\mathcal{H}_1,\mathcal{H}_i}}{d_{\mathcal{H}_1,\mathcal{H}_i}}\right|$ depending on which local minimum is used as the end system's position. We can see that, compared to using the non-global minima, using the global minimum results in a smaller relative error when predicting RTTs.

4.3 Numerical Methods for Finding Local Minima

Finding the maxima or minima of a function is a well researched area of mathematical analysis. For continuous functions with a known first derivative (gradient), identifying local minima is as simple as locating all points where the gradient is equal to zero and evaluating the function value in those points. That way, we can find all minima, maxima and saddle points.

Problems arise when there is no closed form for determining all the points where the gradient of the function is zero. In such cases, numerical methods are used to find local minima (identifying the maximum of a function is equivalent to finding the minimum of the negative of that function). All of those numerical methods initially use one or multiple points in the parameter space as an initial guess, which is considered to be the current

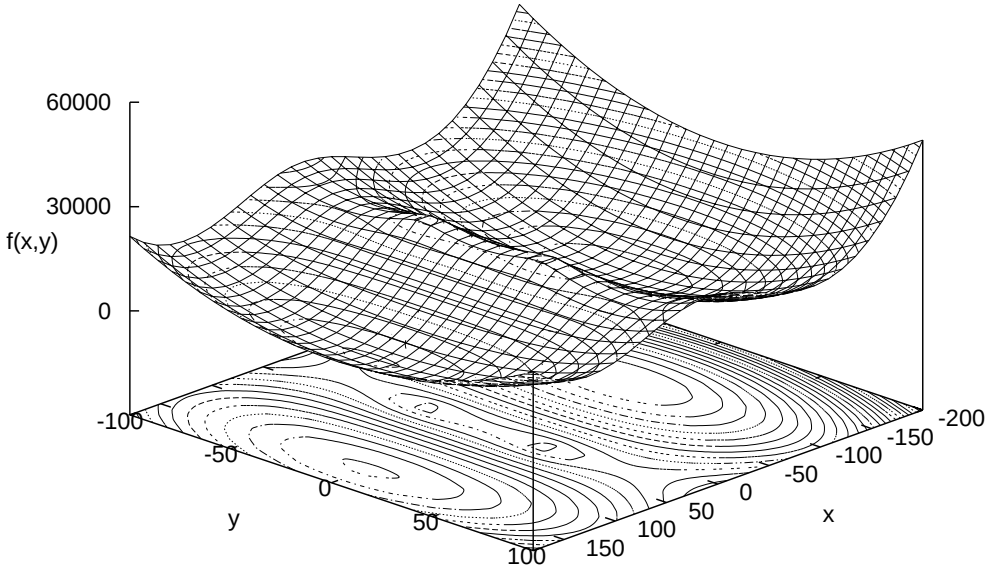

Figure 4.3: Objective Function Demonstrating the Existence of Local Minima in the Case Where Measured Distances are "Ideal".

solution. This current solution is then gradually refined by replacing it with a better one that is obtained by an iterative step. All iterative numerical function minimization strategies also have a terminating condition, i.e., a heuristic which determines if further refinements make sense by considering the computational cost of the preceding step and the level of refinement achieved by it. Usually, the terminating condition is either the distance between the last two solutions in the parameter space or the difference between the function values of the last two refinements. What distinguishes the different numerical function minimization strategies is the method used to refine the solution. In this chapter, we describe the two most popular methods: Downhill Simplex and Conjugate Gradient.

4.3.1 Downhill Simplex

Downhill Simplex [20] function minimization is one of the most general and, hence, most popular function minimization methods. Its popularity is based mostly on the fact that it needs neither first nor second derivative information of the function to perform the minimization.

The Downhill Simplex algorithm works as follows: The starting point of the algorithm is a simplex (the simplest geometric figure which can be constructed in a n-dimensional

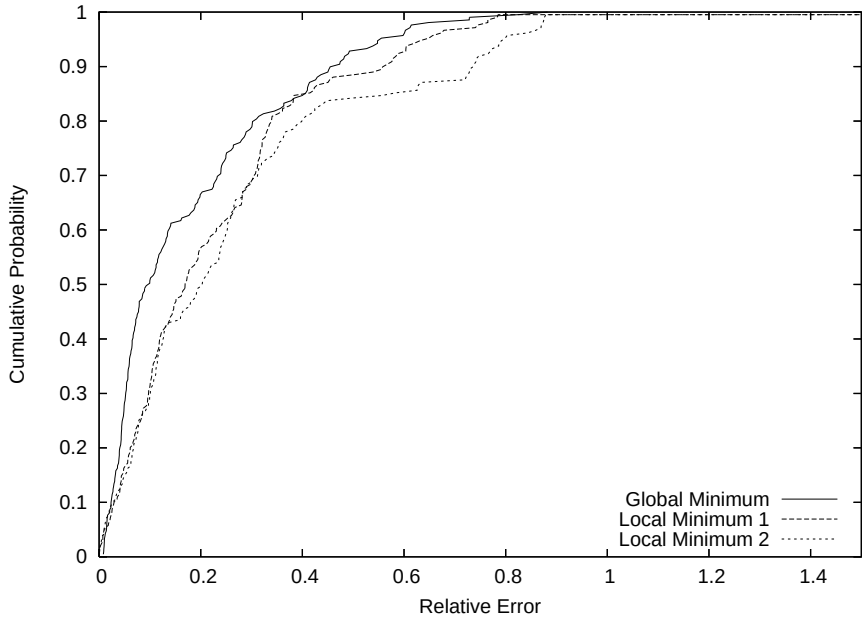

Figure 4.4: CDF of the Relative Error of an RTT Prediction for One End System, Depending on Which of the Three Local Minima Has Been Used

space using $n + 1$ points) defined in the parameter space of the objective function. The points of this simplex must be affine independent (can be used as a base for an n dimensional space) – otherwise the Downhill Simplex method would not be able to minimize the function value in each dimension. Downhill Simplex function minimization always tries to find a better simplex than the current one by moving the "high" point (the point of the simplex for which the value of the objective function is maximal) somewhere into the parameter space. To achieve this, Downhill Simplex applies different transformations to the simplex, such as stretching, contracting or mirroring it. If none of those transformations yields a better point, Downhill Simplex contracts the simplex towards the "low" point of the simplex. The price for the flexibility of the Downhill Simplex method is the rather large number of evaluations of the objective function needed to perform the minimization, since it does not make use of additional information about the objective function such as its gradient.

4.3.2 Gradient Methods

If the gradient of the function can be computed efficiently, gradient methods such as Steepest Descent or Conjugate Gradient can be used to find a local minimum (for a good introduction to different non-linear function minimization methods see [52]). The procedure for all gradient methods is the same: First, a direction for the search is chosen based on the function gradient at the current solution and sometimes based on previous current solu-

tions. After choosing a direction, a one-dimensional minimum search is performed on a line defined by the current solution and the search direction. To perform the one-dimensional function minimization, first the minimum is bracketed by finding a triplet of parameters a, b, c where $a < b < c, f(a) > f(b), f(c) > f(b)$. Then, the bracketed minimum is iteratively reduced, until the difference between a and b is smaller than a pre-defined threshold (Brent's golden section search – for details see [53]). The newly identified minimum is now the new current solution and the procedure for the function minimization is re-iterated until the function value difference of the newly found solution is below the defined threshold.

4.3.3 Other Methods

The two methods discussed above are the most commonly used ones for function minimization. In addition to those, there are numerous other methods, which we did not mentioned, such as Newton's Iterative, Gauss-Newton, different hybrid methods etc. Those methods are not discussed in this chapter, since the two methods mentioned above are sufficient for our purposes. The strategy for the choice of direction is different for every Gradient Method. For example, the Steepest Descent method decides at each iteration to follow the negative of the gradient (the direction, in which the function has the steepest decrease). It has been shown that this is not an optimal strategy in the general case [52] and that other strategies that involve the knowledge of previous local minima, such as Conjugate Gradient, are more efficient.

4.4 Proposed Method for Finding the Global Minimum

Most of the general function minimization methods make very few assumptions about the function they are about to minimize. Generally, this is a good idea, since the method can be applied to a wide range of functions. On the other hand, by exploiting additional information about the function, we can create a less general, but more effective method for finding its global minimum. In this Section, we describe one such method we developed to find the global minimum of the family of functions described in (2.1).

4.4.1 Defining the Boundaries of the Function

As defined in (2.1), the objective function is unbounded. This means that the definition range of the objective function is $\mathbf{R}^n$. Still, there must exist a range where local minima can be found. This range is important, since we are searching for a minimum in one direction within the parameter space and we need to know at which point we can stop searching for the next minimum or maximum. The "range" of the objective function is defined as follows: The objective function is "limited" by a hyper-ball whose center is the center of all landmark positions. The diameter of the hyper-ball is equal to a maximum of all radiuses for each landmark. The radius for a landmark is defined as the sum of the Euclidean distance of the landmark to the center of all landmarks and the double of the measured RTT to that landmark. The rationale behind the choice of this boundary is that it must be easy

to compute and that it includes all possible local minima. Since we define the boundary as a hyper-ball, checking if one point lies within the boundary is as simple as computing the Euclidean distance from the point to the center of all landmarks and comparing this distance with the hyper-ball radius. The hyper-ball radius was chosen in such a way as to ensure that each measured RTT can reach its double length. We made this choice because the positioning of end systems (minimizing the objective function) in the case where the embedding error is not zero, is equivalent to changing the distances to the landmarks. This means that in the process some distances increase while others decrease. Our choice ensures that the boundary allows enough room for stretching the measured distances in the worst case.

4.4.2 Dissecting the Objective Function

The objective function is usually defined as follows:

$$f(\mathcal{C}_\mathcal{H}) := \sum_{i=1}^{m} (d(\mathcal{C}_{\mathcal{L}_i}, \mathcal{C}_\mathcal{H}) - d_{\mathcal{L}_i,\mathcal{H}})^2$$

For each landmark ($\mathcal{L}_i$) there is one summand of the form:

$$f := (d(\mathcal{C}_{\mathcal{L}_i}, \mathcal{C}_\mathcal{H}) - d_{\mathcal{L}_i,\mathcal{H}})^2$$

Represented separately in two dimensions, the shape of this summand looks like in Figure 4.5. As we can see, one summand of the objective function consists of a "peak" at the position of the landmark. The further away we move from the landmark position, the more the function values decrease until a circular valley is reached, which is on the exact distance to the landmark, as the distance measured from the end system. The valley itself is surrounded by a "wall" (a hill of infinite height), which becomes steeper with increasing distance from the landmark. Since the valley surrounding the peak is not just a single point, but a circle, this function has an infinite number of minima, which form a circle whose radius is equal to the measured distance from the landmark to the end system.

Now, if we add two summands (two landmarks) the "landscape" of the function looks like in Figure 4.6. With two summands we can see that the "landscape" has two peaks at the positions of the corresponding landmarks and two valleys. Each valley of this landscape has its minimum on the same "height" (function value), meaning that there are two possible solutions for the position of the end system. To be able to definitively decide where the end system should be positioned, we need the measurement to a third landmark.

In Figure 4.7 we can see the "landscape" of the final objective function with three landmarks. Like in Figures 4.5 and 4.6 we have one "peak" per landmark. Similar to Figure 4.6 we have two valleys (minima), but this time the function value of one of the minima is lower than of the other. The minimum with the lower function value is the minimum we want to find: the global minimum.

Numerical methods for function minimization such as Downhill Simplex or one of the Gradient methods will find either one or the other minimum, depending on the starting

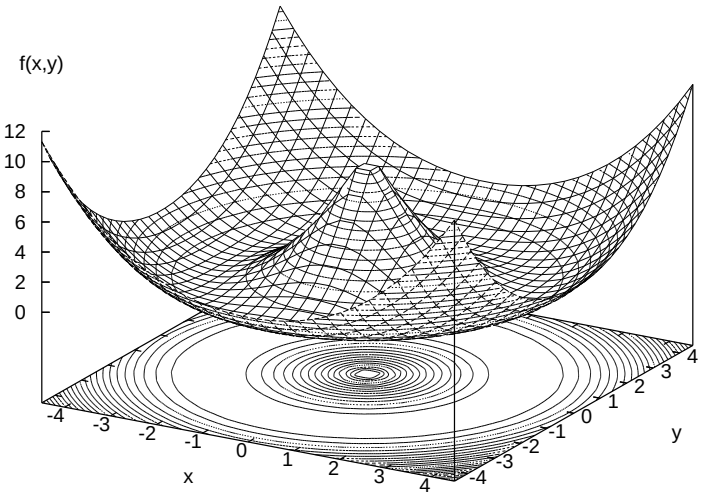

Figure 4.5: Single Summand of the Objective Function Represented in a Two-Dimensional Space.

point. What we try to achieve with our approach is to find most of the local minima. If we can do this, we can also find the global minimum, since one of the local minima must be the global minimum.

4.4.3 Exploring the Function Landscape

In order to find most if not all of the local minima, we propose climbing out of the valley and locating others after the first minimum has been identified. The first question that arises is: How do we know in which direction to climb? To make this decision we have the following information: We know the positions of the "peaks" (landmark positions) and we know our current position. In order to decide in which direction to climb, we should recall the "landscape" defined by two landmarks (shown in Figure 4.6). When we have two landmarks in two dimensions, the function "landscape" has one saddle point which is located between those two landmarks. If we were in one of the valleys in Figure 4.6, the shortest way to reach the other valley would be to climb up the hill and to cross the saddle point between the peaks. We translate this analogy into mathematical language: We should move within the parameter space of the objective function into the direction defined by the position of the current local minimum and the orthogonal to a line drawn trough the positions of the pair of landmarks. Since we do not "know" on which side of the hill we are located, we should try moving in both directions defined by this orthogonal line.

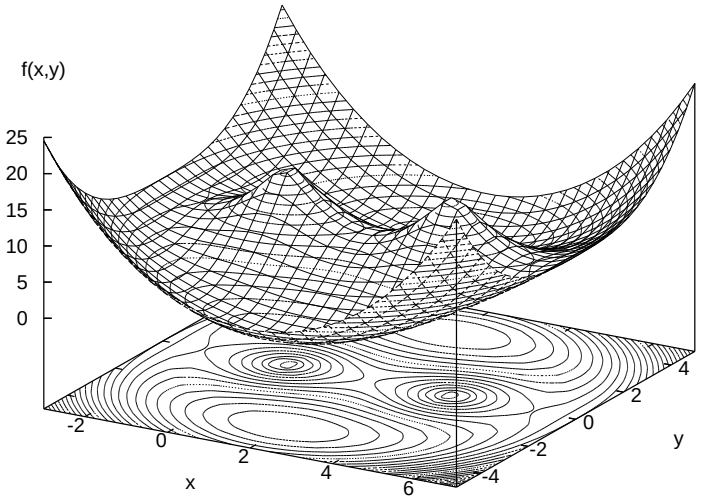

Figure 4.6: Objective Function with Three Landmarks Represented in a Two-Dimensional Space.

4.4.4 Analogy for "Climbing the Hill"

The next question is: What is the mathematical equivalent to climbing a hill? The answer is: We move in one direction in the parameter space (x-y plane in Figures 4.5, 4.6 and 4.7) either until the function stops increasing, or until we have left the "boundaries" of the function. To move in one direction in the parameter space we need a fixed point P and a direction vector dir. Once we have P and dir, we can define "climbing the hill" as finding a parameter $\lambda > 0$, for which the one-dimensional function $f'(\lambda) := f(P + \lambda \cdot dir)$ has a maximum, which is equivalent to finding a minimum of $-f'(\lambda)$. Figure 4.8 shows one such direction. If we are able to find a minimum for $-f'(\lambda)$, we know that we have reached the saddle point (or a point near it) and that the function value of $f'(\lambda)$ decreases beyond this point, meaning that we can now descend into a valley.

4.4.5 Descent Into the Next Valley

After finding the maximum of $f'(\lambda)$, we can now start to descend into the next valley. Since we use a numerical method for finding a maximum of $f'(\lambda)$ we cannot just start a multivariate function minimization in the point $P + \lambda \cdot dir$ since there is the possibility that we would fall back into the starting point of the climb (old valley). To eliminate this possibility, we propose finding the next minimum of $f''(\mu) := f'(\lambda+\mu) = f(P+(\lambda+\mu) \cdot dir)$. Now we can start a gradient-based multivariate function minimization at that point. The result of this function minimization can be either a new minimum (a new valley) or the old minimum, if the mountain we have just crossed leads back to the previous valley. This

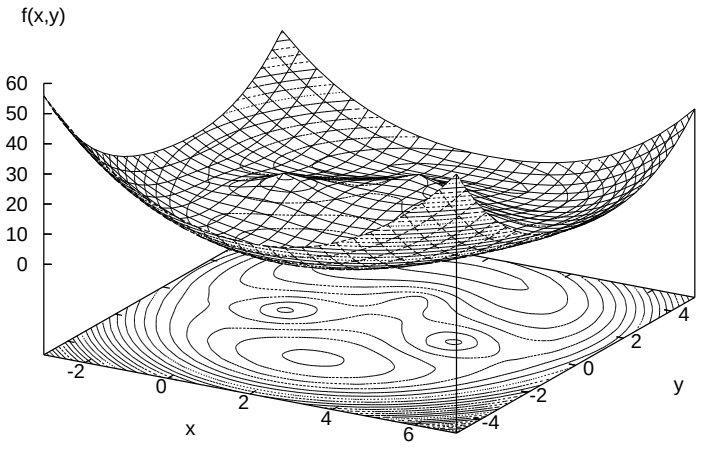

Figure 4.7: Objective Function with Three Landmarks Represented in a Two-Dimensional Space.

procedure can be re-iterated until the "boundary" of the function is reached.

4.4.6 Handling Multiple Dimensions

The analogy we have been using so far is specific to a two-dimensional parameter space. The third dimension, which defines the shape of the landscape in our examples, is used to represent the value of the objective function.

Since RTT prediction schemes based on minimizing the objective function usually gain in precision with an increasing number of dimensions, we need to find a way to identify all valleys in a generalized case of an n-dimensional parameter space. In order to leave the current valley, we need a direction vector. We propose using a vector orthogonal to a hyper-plane defined by n landmarks. In a two-dimensional parameter space, this would be a line orthogonal to a line defined by two landmarks. For three dimensions, the direction would be a line orthogonal to a plane defined by three landmarks, etc. Given the positions of n landmarks $(\mathcal{C}_{\mathcal{L}_1}, \ldots, \mathcal{C}_{\mathcal{L}_n})$ the vector $dir := (x_1, x_2, \ldots, x_n)$ orthogonal to a hyper-plane defined by those landmarks can be computed as the following determinant:

$$\begin{vmatrix} (\mathcal{C}^1_{\mathcal{L}_2} - \mathcal{C}^1_{\mathcal{L}_1}) & \cdots & (\mathcal{C}^1_{\mathcal{L}_{n-1}} - \mathcal{C}^1_{\mathcal{L}_1}) & x_1 \\ (\mathcal{C}^2_{\mathcal{L}_2} - \mathcal{C}^2_{\mathcal{L}_1}) & \cdots & (\mathcal{C}^2_{\mathcal{L}_{n-1}} - \mathcal{C}^2_{\mathcal{L}_1}) & x_2 \\ \vdots & \ddots & \vdots & \vdots \\ (\mathcal{C}^{n-1}_{\mathcal{L}_2} - \mathcal{C}^{n-1}_{\mathcal{L}_1}) & \cdots & (\mathcal{C}^{n-1}_{\mathcal{L}_{n-1}} - \mathcal{C}^{n-1}_{\mathcal{L}_1}) & x_{n-1} \\ (\mathcal{C}^n_{\mathcal{L}_2} - \mathcal{C}^n_{\mathcal{L}_1}) & \cdots & (\mathcal{C}^n_{\mathcal{L}_{n-1}} - \mathcal{C}^n_{\mathcal{L}_1}) & x_n \end{vmatrix}$$

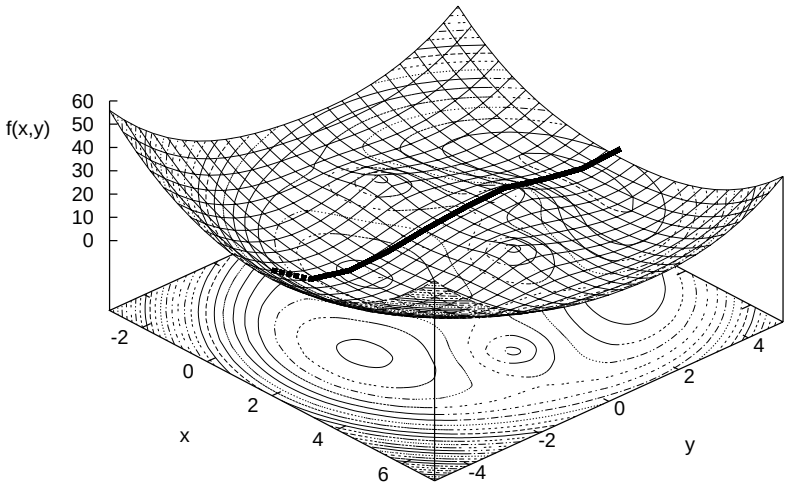

Figure 4.8: One Direction for Leaving the Valley of a Local Minimum.

4.4.7 Algorithm for Exploring the Landscape

In the previous sub-section we have defined everything we need to explore the landscape of the objective function. Now we put all of that together into an algorithm which uses an "educated guess" about the landscape of the objective function in order to find all the valleys in its landscape.

The algorithm uses two sets: a set of starting points (S), which should be used to perform a numerical function minimization and a set of known local minima (M). At the beginning, the set of the known local minima is empty and the set of the starting points contains one starting point, which was randomly chosen or obtained in some other manner. The first step of the algorithm is to remove one point from the set of the starting points and to perform a multilateral function minimization (for example using the Conjugate Gradient method) with that starting point. The result of the function minimization is one local minimum. Then, our algorithm checks whether the found local minimum is in the set of known local minima. If this is not the case, it is added to the set of known local minima and is used to find new starting points.

To find new starting points for the search based on a new local minimum m we use the following procedure: For every subset of landmarks used in the objective function we calculate an orthogonal vector dir (for details see Section 4.4.6). For each such orthogonal vector we try to leave the minimum's valley by finding the next maximum of the one-dimensional function $f'(\lambda) := f(P + \lambda \cdot dir)$ (see Section 4.4.4) and the minimum that is behind that maximum by minimizing $f''(\mu) := f'(\lambda + \mu)$. If we are able to find one such

pair of maximum/minimum, we put that starting point defined as $m' = m+(\lambda+\mu)\cdot dir$ into the set of starting points. We continue our search for the next maximum/minimum pair in the same direction until we leave the "range" of the objective function. The same procedure is also applied in the opposite direction ($dir' := -dir$), since we cannot be sure in which direction the hill really is. After all starting points relative to a new minimum P are found, the new minimum is taken into a set of known local minima and the whole algorithm is re-iterated. The algorithm terminates when the set of the starting points is empty. A more formal version of this algorithm can be found in Algorithm 4.1.

4.4.8 On the Computational Complexity of the Algorithm

The computational complexity of our algorithm (Algorithm 4.1) depends on the following variables: number of dimensions (denoted by n), number of landmarks (denoted by L) and number of local minima of the function (denoted by M). Since it is very difficult to analytically determine the computational overhead of a numerical method for multimodal function minimization, we will only give the best case and worst case estimates.

Considering the number of local minima M, the best case is when the function (2.1) has only one minimum: $M = 1$. In this case, our algorithm performs one function minimization to find that minimum and afterwards tries to "climb" out of the valley of that minimum using different directions. Each direction is calculated based on the positions of n landmarks chosen from the given set of L landmarks. Therefore, the number of possible directions is the number of combinations without repetition: $\binom{L}{n} = \frac{L!}{n!(L-n)!}$. For each such direction we perform two searches: one in the positive and one in the negative direction. Each of these searches results in a certain number of possible starting points for further function minimization.

The worst case for the number of possible starting points is obtained if all landmarks are located on one straight line determined by the search direction and the already found local minimum. In such a case, our algorithm will find $L - 1$ new starting points for the local minimum search. As a result, the number of local minimum searches performed by our algorithm is $\frac{2(L-1)L!}{n!(L-n)!}$. This is of course the upper boundary for the worst case. In the best case no new starting points will be found and hence only one local minimization will be performed.

For the general case where the number of local minima is $M > 1$, we can assume that in the worst case each minimum found will yield $\frac{2(L-1)L!}{n!(L-n)!}$ starting points for the local search. This gives us the upper boundary of the worst case: $\frac{2M(L-1)L!}{n!(L-n)!}$. In the best case each minimum will yield only one new starting point, in which case the number of performed local minimum searches equals M.

It is obvious that our algorithm can yield a large number of local function minimizations. Especially in the case where the number of dimensions of the virtual space n is low and the number of landmarks L is high, our algorithm will perform a very large amount of local searches. On the other hand, with a decreasing number of dimensions, the computational overhead for finding the local minimum also decreases. Also, compared to other global minimum search algorithms, our algorithm guarantees to terminate in a finite number

Algorithm 4.1 Algorithm for enlisting all local minima

Require: LMS set of landmark positions
Require: DST vector of distances measured to landmarks
Require: D number of dimensions
  $S \leftarrow$ {random point} {Start with a random point}
  $M \leftarrow \emptyset$ {Start with an empty set of local minima}
  for $p \in S$ **do**
    $m \leftarrow find\_min(p, LMS, DST, D)$ {Find a local minimum}
    if $m \notin M$ **then**
      for $os \in \{x | x \in LMS, \|x\| = D\}$ **do**
        $dir \leftarrow orthogonal(os)$ {Calculate the orthogonal vector to landmark positions}
        $\lambda \leftarrow 0$
        while $m + \lambda \cdot dir$ within function boundaries **do**
          $\lambda \leftarrow next\_max(\lambda, m, dir)$ {Find next one-dimensional function maximum relative to λ starting at m along the direction vector dir}
          $\lambda \leftarrow next\_min(\lambda, m, dir)$ {Find next one-dimensional function minimum relative to λ starting at m along the direction vector dir}
          if still inside function boundaries **then**
            $S \leftarrow S \cup \{m + \lambda \cdot dir\}$
          end if
        end while
        $\lambda \leftarrow 0$ {Perform search also in the negative direction}
        while $m + \lambda \cdot -dir$ within function boundaries **do**
          $\lambda \leftarrow next\_max(\lambda, m, -dir)$ {Find next one-dimensional function maximum relative to λ starting at m along the direction vector dir}
          $\lambda \leftarrow next\_min(\lambda, m, -dir)$ {Find next one-dimensional function minimum relative to λ starting at m along the direction vector dir}
          if still inside function boundaries **then**
            $S \leftarrow S \cup \{m + \lambda \cdot -dir\}$
          end if
        end while
      end for
    end if
  end for
  return S

of steps, assuming the non-pathological case of (2.1), where the number of local minima M is finite. This is also an advantage to other random start point methods, which lack a good terminating criterion.

4.5 Evaluation

To verify if our proposed algorithm increases the probability of finding the global minimum we used the following experiment. We compared the results of the standard GNP end system positioning system in terms of probability to find the global minimum using the same data used to show the existence of multiple minima in the objective function from Section 4.4.

In order to have comparable results, we implemented the end system positioning from GNP using Downhill Simplex function minimization provided by [52]. We also implemented our algorithm using Conjugate Gradient, linear minimization (golden section search with hyperbolic extrapolation) and one-dimensional minimum bracketing functions provided by the same source [52]. For the experiment we used a fixed number of landmarks (10) and varied the number of dimensions between 2 and 8. To have comparable results and to avoid interference when using different landmark positions, we based all our objective functions on the same landmark coordinates. The landmark coordinates were calculated only once (per number of dimensions) and used by both our and the GNP algorithm. For each end system from our RTT distance matrix we calculated the end system coordinates using the GNP algorithm and our algorithm. We also estimated the total number of local minima of the objective function by performing Conjugate Gradient minimum search with 200 different starting points.

Figure 4.9 shows the percentage of global minima found by GNP and by our algorithm. As we can see, our algorithm correctly identified the global minimum of the objective function for almost each end system in our sample. However, there are some cases, where our algorithm is unable to find the global minimum. The reason for this is that our algorithm uses only a heuristic to minimize the probability of not finding the global minimum. It does not guarantee finding *every* minimum there is.

4.6 Conclusion

In this chapter, we identified the problem of the existence of multiple local minima in the objective function used to position end systems in a virtual space. The existence of multiple local minima has an impact on our perspective of minimizing a function. It is not enough to have a method that is fast and precise in finding a local minimum. We also need an algorithm that is able to find the global minimum of the objective function. Our research has shown that the existence of multiple minima in the objective function occurs too frequently than could be ignored.

To overcome this problem we developed an algorithm which exploits the knowledge of the objective function's "landscape" when trying to find all local minima and thus the

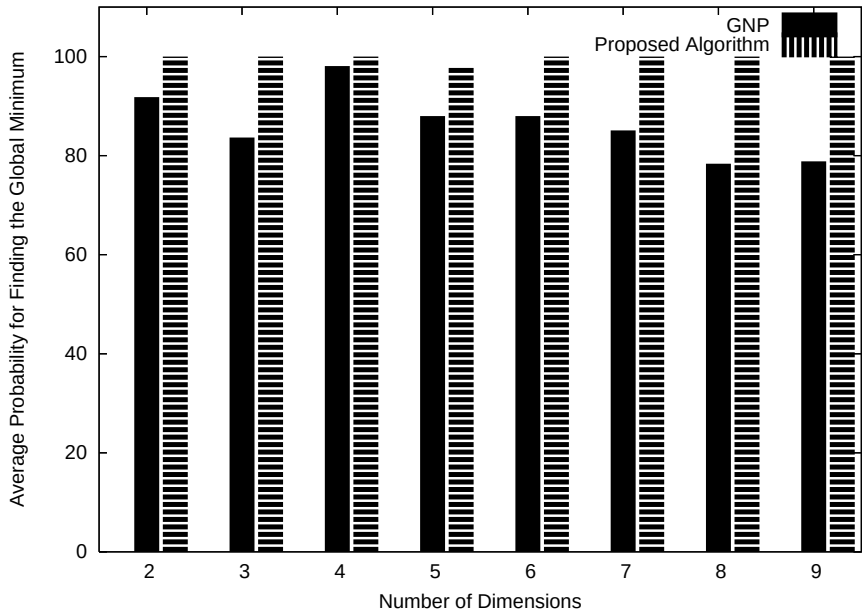

Figure 4.9: Comparison of the Probability for Finding Global Minima Using GNP and our Proposed Algorithm.

global minimum. Our evaluation has shown that our algorithm performs far better than the plain end system positioning by function minimization as proposed by GNP.

So far we have considered optimizations of GNP. Our assumption was that in order to create a topology-aware overlay network, we need an embedding in a virtual space. In the next chapter, we will introduce a scheme that is able to construct a topology-aware overlay network without the need of embedding all end systems into a virtual space beforehand. After that, (in chapter 6), we will use this overlay network to embed end system into a virtual space in order to obtain a better prediction scheme.

Chapter 5

Fisheye Overlay Network

5.1 Introduction

In chapters 3 and 4 we assumed that, in order to be able to construct a topology-aware overlay network, an embedding of end systems into a virtual metric space is necessary. In this chapter, we show that this is not the case. In this chapter, we introduce a scheme we invented [54] for constructing a topology-aware overlay network without beforehand embedding end systems into a virtual space.

As of today, there are numerous p2p and overlay protocols. Most of them are designed to provide services such as service discovery [55, 56], DHT [6–8, 35, 57], application-layer multicast (ALM) [36–38]. All of those protocols are designed to be scalable and efficient within the overlay network. For example, the number of hops for a lookup in Chord [6] is guaranteed to grow logarithmically with the size of the network. On the other hand, the same protocol totally disregards the topology of the underlying network. This means that routing from end system A to end system B that are in the same autonomous system (AS), may be performed via an end system C that is in another AS. As a result, one query may unnecessarily traverse "long" (in terms of communication delay) links in the underlying network several times before it reaches its destination. This unnecessarily increases the communication delay.

As described in 1.5.5, we can use either use measured RTTs or a prediction scheme such es embedding in a virtual metric space to obtain topology information of the underlying network. We could use this information in order to construct a topology-aware overlay network.

The problem is that performing RTT measurements for every pair of end systems in an overlay network takes some time. At the same time the overhead of storing such information increases quadratically with increasing number of end systems participating in the overlay network. Finding an optimal solution is also not easy. In fact, it is NP-hard. This makes finding the optimal solution not practical even for moderate sized overlay networks.

In order to obtain a usable method to build a topology-aware overlay network we need to reduce the number of required measurements and the overhead of overlay optimization. To have scalable measurements, we could reduce the number of measurements performed

at the cost of accuracy by eliminating the need to perform measurements for all end system pairs. Another effective method is to have a system that does not need the RTT information for every pair of end systems immediately. Instead, the system can perform a constant number of measurements per time unit and use this newly obtained information to continuously converge towards a better solution.

Numerous position-based mobile ad-hoc network routing protocols [54, 58] are based on a "fisheye" view of the network. The "fisheye" view of the network implies that each end system has a view of its neighborhood similar to how a fisheye sees. The lens of a fisheye has an extreme wide angle. The light is refracted by the fisheye lens such that the center of the view contains very high level of detail. With increasing distance from the center of the view, the detail level rapidly decreases.

In this chapter, we propose a black box based approach to build an overlay network. We propose periodically measuring RTTs and using that information to build a fisheye view of the overlay network at each end system. This fisheye view is constantly refined when new information is available. By doing so we ensure constant convergence towards a better overall solution for the overlay network. In our approach we combine the ideas of having a bidirectionally connected graph that is continuously improved. This perpetual refinement of the overlay network resembles the approach of the mesh refinement proposed in NARADA. We also build a fisheye view of the overlay network at each end system as proposed by Meridian. We improve Meridian's method for choosing neighbors by minimizing the forces of gravity instead of having concentric rings and maximizing k-polytopes.

This chapter is structured as follows. In Section 5.2 we present how the fisheye view paradigm can be applied to computer networks. In the same Section we show how a fisheye view can be incrementally built for overlay networks even without using an embedding of end systems in a virtual space. Section 5.3 identifies the problems that occur, if we use a naive approach of independently building a fisheye view at each end system. Our solutions to the identified problems are presented in the same Section. Section 5.4 presents a distributed communication protocol used to incrementally build and refine the fisheye view of the overlay network on each end system. Section 5.5 compares our approach with other existing topology-aware approaches. In the last section we summarize our approach, the experimental results and outline the possible use of such an overlay network.

5.2 Fisheye View

When we apply the paradigm of a fisheye view to a position-based computer network, we assign to each end system a fisheye view of the overlay network. This fisheye view is populated with many near neighbors. The density of known neighbors decreases with increasing distance. Theoretically, the density should never reach zero. In the case of an overlay network this is not possible, since both the number of end systems and the diameter of the space populated by the end systems are finite.

The advantages of the fisheye view at each end system of the overlay network are twofold. The fisheye view decreases the amount of data that one end system must store

about the network. Since memory at each system is limited, the amount of information stored should be kept constant regardless of the overlay network size. A fisheye view also contains a rather high density of "close" neighbors allowing efficient routing using short distances. At the same time, it contains minimal information about remote neighbors, making the long routing "jumps" possible.

The fisheye view paradigm is not only limited to network routing protocols. Many of the existing DHT approaches are based on a fisheye routing table to achieve a balance between the size of the routing table and the number of routing hops. For example, Pastry [7] and other Plaxton-based routing protocols [35] use prefix-based routing table, which is nothing else than a fisheye view of the ID space.

Almost all existing fisheye view approaches [6, 7, 54, 57, 58] are based on an assumption that each end system has a position in a one- or a two-dimensional Euclidean space. When we consider a generic computer network such as the Internet, we are dealing with a connected graph. There is no guarantee that this graph can be embedded into a metric space. Even if there is such an embedding, there is no guarantee that the number of the dimensions used for the embedding is low. To overcome this problem, we could use an approach that does not necessarily embed the distances, but tries to embed distance approximations in order to reduce the overall error. For example, we could use some kind of position-based RTT prediction scheme such as GNP [15, 17] or VIVALDI [16] to obtain coordinates of the end systems. The problem of such an approach is that they are per design not accurate – mostly due to the fact that RTTs in the Internet frequently violate one of the basic properties of metric distances: the triangle inequality. Nevertheless, for the clarity of our explanations, we assume that there is an accurate embedding of end systems into a k-dimensional Euclidean space, in such a way that the distance between any end system pair in the metric space is exactly the measured RTT between those end systems. Later, we will show that there is actually no need to perform the embedding, since we can operate only with measured distances, which significantly simplifies our approach.

To explain our idea for building a fisheye overlay network we assume the following. Our end systems in the overlay network are defined as a set of k nodes $\mathcal{N} := \{n_i | i \in \{0 \dots k\}\}$. Each of the nodes has a position denoted by $\mathcal{C}_{n_i}$ in a k-dimensional Euclidean space denoted $\mathcal{S}_k$. The positions of the nodes in $\mathcal{S}_p$ are not random – they are chosen in such a way that for every pair of nodes $n_l, n_m \in \mathcal{N}$ the Euclidean distance, defined in (5.1)

$$\hat{d}(\mathcal{C}_{n_l}, \mathcal{C}_{n_m}) := \sqrt{\sum_{i=1}^{k} (\mathcal{C}^i_{n_l} - \mathcal{C}^i_{n_m})^2} \tag{5.1}$$

is equal to the RTT between those two nodes measured through the underlying network $d(n_l, n_m)$. Each node is allowed to store information of at most c neighbors at the same time. The problem we are trying to solve is to assign a fisheye view choice of neighbors to each end system in a distributed manner. This choice should fulfill the properties of the fisheye view, meaning that most of the neighbors should be from the direct vicinity in terms of network distance (RTT). At the same time, the fisheye view should also contain a few "long" links to neighbors that are more distant. Those long links enable efficient routing

towards end systems that are far away. Another important property of a fisheye view is the geographical diversity. Geographical diversity means that if we embed the overlay network into a metric space, the chosen neighbors would "surround" the end system. The geographical diversity ensures efficient routing in each "direction" through the overlay network.

5.2.1 Universe Analogy

In order to explain our approach we use the universe analogy. We consider an overlay network to be a universe. Each end system is one planet located somewhere within this universe. Distances between the planets are RTTs measured between the corresponding end systems. Since the universe has no reference point, each planet has its own "view" of the universe. Each planet's view considers this planet to be in the center of the view. The problem we are trying to solve is to reduce the number of planets within the view to a constant number c. This reduction of the planet's view is equal to choosing the overlay neighbors for one end system. The two most distinguishing properties of the fisheye view are geographical diversity and a degree of detail that decreases with increasing distance to the center of the visual field. Therefore, the reduction we perform should consider those properties. As in the real universe, there is the gravity force in our universe. This gravity force is defined by (5.2), where r is a distance between two planets and m_1 and m_2 are masses of those two planets.

$$F = G \cdot \frac{m_1 \cdot m_2}{r^2} \tag{5.2}$$

5.2.2 Geographical Diversity

Let us assume, we would only try to achieve geographical diversity. As mentioned before, the planet for which we tailor its view is in the center of the view. We denote this planet (end system) with n_c. The other planets surrounding it are not necessarily uniformly distributed through the universe. There may be some clusters of planets that are located close to each other. Also, there may be some portions of the universe containing no planets at all. This means that the density of planets is not evenly distributed through the universe. The goal of our reduction of planets in the view is to remove the planets that are close to each other, i.e., to avoid clusters. We could achieve this by iteratively removing planets, one by one. At each iteration we would have to pick one planet that is in the dense part of the universe. This procedure can be continued until we have exactly c planets left. The planets that are still in the view are evenly distributed through the universe and therefore the obtained view is geographically diverse.

The only question that remains is: Which planet should be removed at the iteration? As an answer to this question, we propose removing the planet with the largest sum of gravity forces, because those planets are in the most dense parts of the universe. We assume that each planet in the universe has a constant mass (for simplicity we assume $m = 1$). For each planet pair n_i, n_j within the universe we could calculate the gravity force using (5.2). This gravity force increases (more than linearly) with decreasing distance between planets. For that reason, the total sum of gravity forces for planets that are in a cluster of planets

is higher than for planets that are located in less dense parts of the universe. Therefore, if we always remove the planet with the maximum sum of gravity forces from the view, we will equalize the density of planets. The equation for computing the gravity force would be (5.3).

$$F_{n_i^c} = \sum_{j \in \mathcal{N} \setminus \{i,c\}} \frac{1}{(d(n_i, n_j))^2} \tag{5.3}$$

5.2.3 Shaping the Fisheye View

Up to now we assumed that the mass of a planet is equal for every planet. As a result of this, the reduced view of planets is (as far as possible) equally distributed. This equal distribution of the view unfortunately does not have the fisheye property. In order to have the fisheye property, we should favor planets close to the center of the universe (the planet for which we calculate the view). To do so, we propose considering the mass of planets not to be constant, but proportional to the distance to the center of the universe n_c. By doing so, we shift the density of the planets remaining in the view towards the center of the universe to obtain a fisheye view. The final equation for computing the total gravity force for one planet n_i for the center n_c is (5.4).

$$F_{n_i^c} = \sum_{j \in \mathcal{N} \setminus \{i,c\}} \frac{d(n_i, n_c) \cdot d(n_j, n_c)}{(d(n_i, n_j))^2} \tag{5.4}$$

As an illustration, Figure 5.1 shows the result of our algorithm applied to a "universe" of 300 end systems. The universe is reduced to 25 neighbors. Of course, in a p2p network, one peer never stores the information about the whole overlay network at the same time. Instead, one peer becomes aware of only few new peers at a time. Fortunately, our algorithm is by design incremental and can be applied each time a new peer is known, to determine whether the new peer should replace an existing one in the view. For example, if one peer already has its fisheye view of the network and becomes aware of a new peer, it only needs to add this peer to its "view" that now contains the old fisheye view and additionally the new peer, and to re-calculate the sum of the gravity forces in it. If the new "planet" has a lower sum of gravity forces than the other planets, then the planet with the largest sum of gravity forces is removed from the fisheye and the new neighbor becomes a new planet in the "view". By this an end system incrementally refines its fisheye view through its lifetime.

5.2.4 Are Positions Required?

If we examine (5.4), we see that the total gravity force depend only on distances. Since in our case the distances are measured RTTs, we could use our algorithm without embedding the end systems into a virtual space. Instead of embedding, we can directly use the measured distances (RTTs) to construct the fisheye view for each end system. Since our approach does not require the knowledge of the host position, it avoids the error introduced by performing the embedding. This means that our approach should still be effective, even

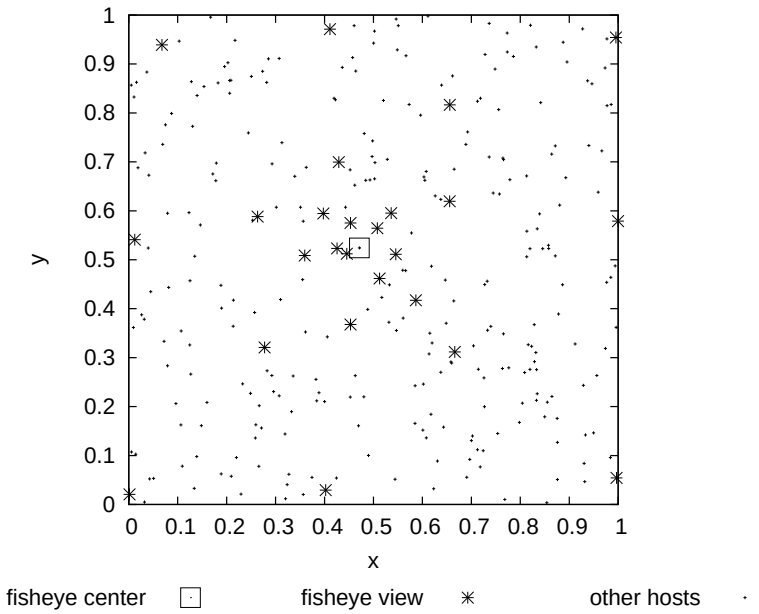

Figure 5.1: Example of fisheye view calculated using our gravity force minimization algorithm.

if not all distances fulfill the triangle inequality property. This has been substantiated by experiments performed in [14–17] that show that those embeddings provide a quite precise RTT prediction service. This means that assuming that the Internet can be embedded in a metric space is not completely false.

5.3 Problems of Independent Construction of the Fisheye View

If we simply apply the method we described in Section 5.2, each end system in the overlay network would be able to construct its own fisheye view of the network. At the first glance, this would be a good idea, since there is no requirement for cooperation between the end systems, besides providing RTT measurements. But there are also two major drawbacks to this approach: asymmetry of the connection graph and non-uniform connection distribution.

5.3.1 Asymmetry of the Connection Graph

We can observe an overlay network as a connected graph. If each end system independently decides which other end systems are known to him, we obtain a directed graph. Such a graph has no guarantee that it will be connected. For example, if we have a portion of the virtual space with high end system density, the chances are growing that at least one of those end systems will not be “chosen” by some other end system to be its neighbor. As a consequence, if one of the end systems is never chosen as a neighbor, there is no edge in

the graph representing the overlay network that leads to this end system. This means that the graph is not necessarily connected.

To overcome this problem we propose that the "is a neighbor of" relation of a fisheye view is bidirectional. This means n_1 is a neighbor of n_2 iff n_2 is a neighbor of n_1. If the "is a neighbor of" property holds, the graph is bidirectional and must be connected. This is due to the fact how the overlay network is built. We assume that the existing overlay network is a connected graph. If one node joins the network, the new node must connect to at least one of the end systems of the existing network. If this connection is bi-directional, the graph of the extended overlay network is also connected.

5.3.2 Non-Uniform Connection Distribution

Another problem that can occur, if the end systems are not uniformly distributed within the virtual space, is the non-uniform distribution of connections in the overlay network. The problem occur if one end system is isolated and alone in a large portion of the virtual space. Since this end system is the only end system in a quite large portion of the space, the total sum of the gravitation forces for that end system is probably quite low. As a consequence, this end system will be in the fisheye view of many other end systems. Thus this end system may be forced, depending on what the overlay network is used for, to handle a lot of traffic.

To solve this problem, we split the fisheye view into incoming and outgoing connections. Each end system can have at most $\mathcal{C}^{\text{inc}}$ incoming and $\mathcal{C}^{\text{out}}$ outgoing connections. An outgoing connection is a connection initiated by the local system. Incoming connection means that the connection is initiated by some other end system. Each end system is free to use its outgoing connections to improve its fisheye view by disconnecting from one and connecting to another remote system, if that improves its fisheye. However, an end system is not allowed to disconnect incoming connections. We also add a multiplication factor for the total sum of gravity forces (5.4) to stimulate connecting to the end systems with lower number of incoming connections: $\frac{1}{\mathcal{C}^{\text{inc}}-\mathcal{E}^{\text{inc}}}$, where $\mathcal{E}^{\text{inc}}$ is the current number of open incoming connections. Thus, the improved function to compute the sum of gravity force in the universe is (5.5).

$$F_{n_i^c} = \sum_{j \in \mathcal{N} \setminus \{i,c\}} \frac{1}{\mathcal{C}_j^{\text{inc}} - \mathcal{E}_j^{\text{inc}}} \cdot \frac{d(n_i, n_c) \cdot d(n_j, n_c)}{(d(n_i, n_j))^2} \tag{5.5}$$

5.4 Network Protocol for Construction of a Distributed Fisheye View

In this Section we propose a communication protocol for building a fisheye overlay network. Our protocol is designed to incrementally converge towards the optimal solution by using only partial measurement information and knowledge of the overlay network.

The goal of the algorithm and the protocol is to build and maintain a fisheye view at each end system. The constructed overlay network represents a bidirectionally connected graph, where each end system has knowledge of only a constant number of other end systems. The

protocol is constructed in a way such that it is robust against end node failures. Also it is reactive to topology changes of the underlying network.

5.4.1 Gossiping and Keep-Alive Messages

Each end system participating in the overlay network sends periodically *UPDATE* messages to all of its neighbors. To avoid synchronous "heartbeats" of the network, the retransmission time is randomly (with uniform distribution) chosen from the interval $t_{\text{xmit}} \in \left[\frac{t_{\text{update}}}{2}, t_{\text{update}}\right]$. The *UPDATE* message contains the current fisheye view of the end system and a gossiping list. One purpose of the *UPDATE* message is to confirm the aliveness of an end system to all of its neighbors (keep-alive message). At the same time, the *UPDATE* message performs a random gossiping of new end systems that can be used to optimize fisheye views of the neighbors.

5.4.2 Opening and Closing Connections

Each end system may at any time open a new outgoing connection to any other end system in the overlay network. Usually one end system contains a list of end systems that should be used to optimize the overlay network. This list is maintained by combining information from *UPDATE* messages received from the neighbors. Opening the connection is initiated by sending an *OPEN* message to the remote end system. Upon receiving an *OPEN* message, the end system may either reply with an *ACCEPT* or with a *REJECT* message. The response of the end system depends on the number of already established incoming connections. If the number of already established incoming connections reached its maximum, then the response will be *REJECT*. Otherwise the response is *ACCEPT*. With an *OPEN* message, the list of all current neighbors is sent. If the remote system replies with an *ACCEPT* message, it also must perform RTT measurements to all end systems from the list contained in the *OPEN* message and send them back as a part of the *ACCEPT* message. This information is used at the connection initiating end system to calculate a new fisheye view. If the new fisheye view represents a better fisheye view, in terms of the force of gravity minimization described in Section 5.2, then the initiating end system sends a *COMMIT* message to the remote system. Otherwise, a *ROLLBACK* is sent to the remote system, to indicate that the connection has not been established.

The closing of an established outgoing connection is indicated by sending a *CLOSE* message to the other end system. Also, an end system gracefully leaving the overlay network, should send a *CLOSE* message to all of its neighbors, regardless whether the connection is incoming or outgoing.

5.4.3 Bootstrapping

To join a fisheye overlay network, an end system must "know" at least one node already joined to the overlay network. To find this node, one of the usual p2p bootstrapping methods can be used, such as having a well known peer, dynamic DNS, etc. [59]. The new node

contacts its bootstrapping node by sending a *HELLO* message to it. As a response to a *HELLO* message the bootstrapping node sends the *UPDATE* message that contains its current neighbors and a gossiping list. The gossiping list contains a random choice of known systems that are not the direct neighbors. Each end system maintains a list of such potential neighbors. The reason for this is to increase the random choice of neighbors to connect to. Since this is the same message used for maintaining and gossiping in the overlay network, the joining node can use the information contained in it to choose its neighbors for its initial fisheye view and establish a connection as described in Section 5.4.2.

5.5 Evaluation

In order to verify the performance of our approach, we have compared the overlay network constructed using our approach with binning, an optimal heuristic and a random overlay construction approach [34]. Since we are focusing on the properties of the constructed overlay network, we avoid using any particular overlay routing protocol. Instead, we always calculate an optimal path in the overlay network (path with the minimal RTT stretch) and compare the lengths of those paths.

5.5.1 Protocol Simulation

We evaluated our overlay construction approach using an event based simulation [60]. In order to have a realistic network model, we used RTT information measured in the Internet. In our network model, each overlay message was delayed for half of the RTT measured between end systems in the Internet. The RTT distances are obtained by performing RTT measurements for each pair of end systems in the Internet. In our case, we have used one RTT matrix obtained from the Planet Lab "all sites ping" experiment [41] and one obtained using the King [42] method. Used data sets contain a complete RTT distance matrix for 218 Planet Lab end systems and 462 end systems measured with the King method. The nodes in our simulation are started with 0.1 second delay, meaning a high churn at the first 21.8 seconds of simulation in the case of Planet Lab data and 46.2 seconds for the King data set.

5.5.2 Comparison Methodology

The goal of our evaluation is to determine, whether the overlay network constructed using our fisheye view (*FISHEYE*) approach is better or worse than existing approaches. We compare our approach with the following other approaches:

- *RND:* The random approach is the most common one used in unstructured overlay networks. Each end system randomly chooses S other end systems already participating in the overlay network. The performance of this approach is an upper boundary for a topology-aware overlay network, since it totally disregards the topology of the underlying network.

- *OPT-HEUR:* A heuristic method that should approximate the optimal choice of neighbors. This approach is based on choosing $S/2$ nearest neighbors of an end system and $S/2$ of randomly chosen neighbors from the overlay network. According to [34], this approach delivers results that approximate an optimal solution. We used this heuristic method as a lower boundary for the performance of an overlay building strategy, since finding an optimal overlay network is NP-hard. The main drawback of this method is that each end system would need to have a complete knowledge of the overlay network in order to determine the $c/2$ nearest neighbors.

- *BINN:* In a topology-aware approach [34]. Each end system is assigned into one of the $L!$ bins relative to L randomly chosen landmarks. A bin is determined by measuring RTT to a set of landmarks and ordering those landmarks according to the relative distance to an end system. End systems in the same bin are considered to be roughly in the same area of the network. Hence, similar to *OPT-HEUR*, this strategy assigns $S/2$ neighbors from the same (or the most similar) bin and $S/2$ neighbors chosen randomly from the overlay network. For our experiments, we have set the number of landmarks $L = 5$, which gives the maximal number of 120 different bins

To compare the efficiency of the paths in the overlay network, we compare the relative length of the path through the overlay network in terms of RTT between each pair of end systems in the overlay network. We define the relative path between two overlay systems as:

$$\frac{\text{RTT of the optimal path through the overlay network}}{\text{RTT of the optimal path through the underlying network}}$$

The relative path length of 1.0 means that the communication delay through the overlay network is optimal (as good as the best route through the underlying network). A relative path length value larger than 1.0 indicates the "penalty" (additional overhead) for routing using the overlay network. Values smaller than 1.0 are not possible.

5.5.3 Relative Path Length

Since the performance of all approaches varies depending on the initial random seed, we performed 10 runs of each experiment with different seeds. Figures 5.2 and 5.3 show the median of relative paths for the overlay networks using the Planet Lab and King data sets for all four approaches depending on the number of neighbors (S) that we varied between 9 and 21. As the Figures show, the median of all relative paths in our approach outperforms both RND and BINN overlay construction strategies and approaches the performance of *OPT-HEUR* for increasing S.

In order to keep Figures 5.2 and 5.3 readable, we omitted confidence intervals. To have a better overview of the performance, we also show the cumulative distribution function (CDF) of optimal paths for 9 and 21 neighbors in Figures 5.4 and 5.6 for Planet Lab and in Figures 5.5 and 5.7 for the King data set.

The CDF data for 7 neighbors show that *FISHEYE* and *BINN* do not significantly differ. With increasing number of neighbors, performance of *FISHEYE* improves and for 21

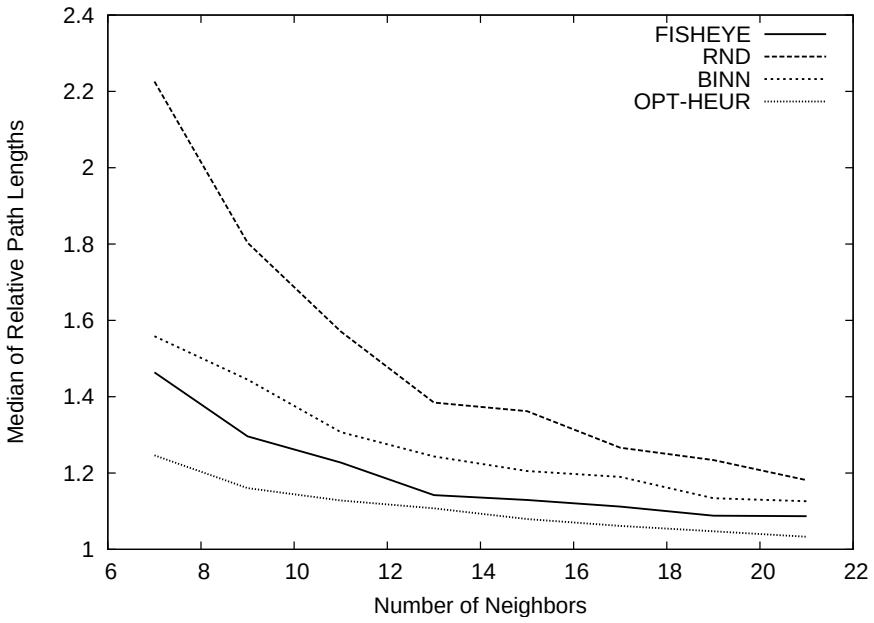

Figure 5.2: Median of relative path length in the overlay network in dependence of number of neighbors for the Planet Lab data set.

neighbors outperforms the *HEUR-OPT* strategy in the King data set. Both median and CDF comparisons show that the FISHEYE approach clearly outperforms both RND and BINN overlay construction strategies for a number of neighbors larger than 7. Another interesting result is that with the increasing number of neighbors, the *FISHEYE* approach converges towards the performance of the OPT-HEUR approach.

5.5.4 Convergence

We also examined the speed of convergence of our approach at high rates of joins and leaves. For this, we simulated building a *FISHEYE* overlay network with $S = 19$ using the King data set for 500 seconds. Every 10 seconds of the simulation, we calculated optimal paths in the constructed overlay network. The median of this data is shown in Figure 5.8. Since we used the King data set for this experiment, all end systems have joined the overlay network after 46.2 seconds of simulation time. The system converged towards a stable state at 210 seconds after the start of the simulation. This means that the overlay protocol reaches a stable state after roughly 3.5 times the duration of a high churn period.

5.5.5 Comparison With Pastry

As another illustration of efficiency of our overlay building protocol, we compare it with the open source implementation of the Pastry protocol [61]. We compared the relative lengths of optimal paths in the fisheye overlay network and the path lengths using the Pastry routing.

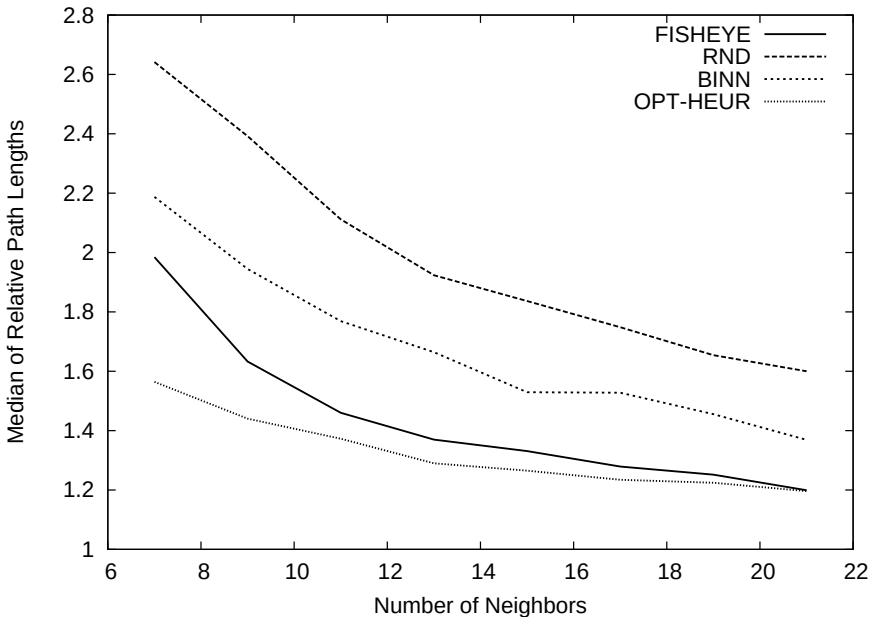

Figure 5.3: Median of relative path length in the overlay network in dependence of number of neighbors for the King data set.

For this comparison, we used the Planet Lab data set. For our approach, we used $S = 18$ (18 neighbors). Although Pastry is topology-aware and has much larger routing tables (potentially more than 100 neighbors), Figure 5.9 shows that after the stabilization phase, our approach outperforms the Pastry overlay by a factor of 7.

5.6 Conclusion

In this chapter we have presented a fully distributed method for topology-aware selection of neighbors in order to construct an overlay network. This method is based on providing a fisheye view of the overlay network to each end system participating in it. The basic principle of the proposed method is to incrementally refine the fisheye view of each end system.

The constructed overlay network is guaranteed to be a fully connected bidirectional graph and has a constant upper limit of known neighbors (limited maximal fan out). Those guarantees make such an overlay network a perfect candidate for numerous overlay applications such as topology-aware multicast service or as the overlay network for an unstructured P2P network. In the evaluation we showed that the overlay network constructed using our method outperforms the one constructed by an already existing binning overlay building strategy in terms of relative path length of routes.

In this chapter we showed that it is possible to construct a topology-aware overlay network. The presented approach is both scalable adaptive to changing underlying network

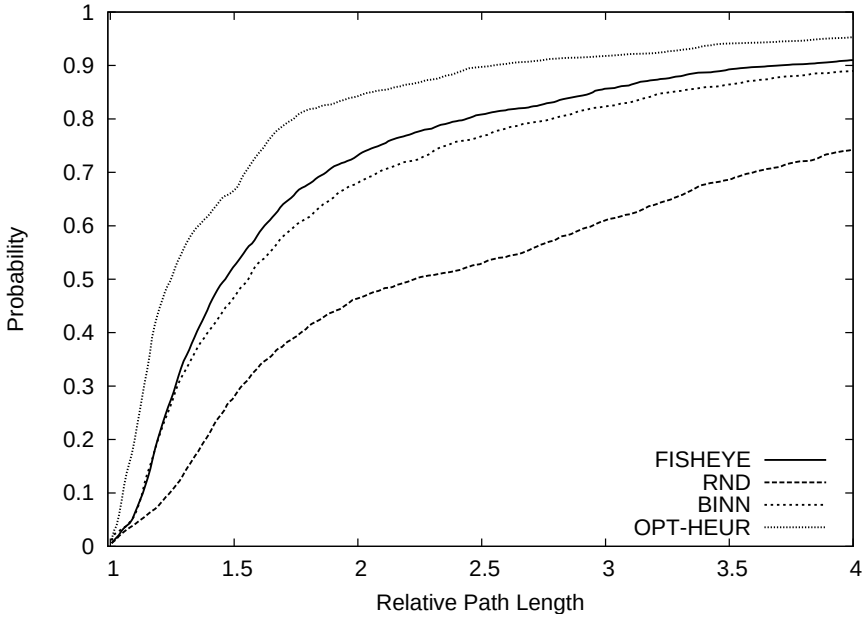

Figure 5.4: CDF of relative path lengths in the overlay network with 7 neighbors for the Planet Lab data set.

topology. In next chapter, we will use this overlay network for improvement VIVALDI decentral scheme for embedding end systems in metric spaces. We named this new embedding system NetICE9.

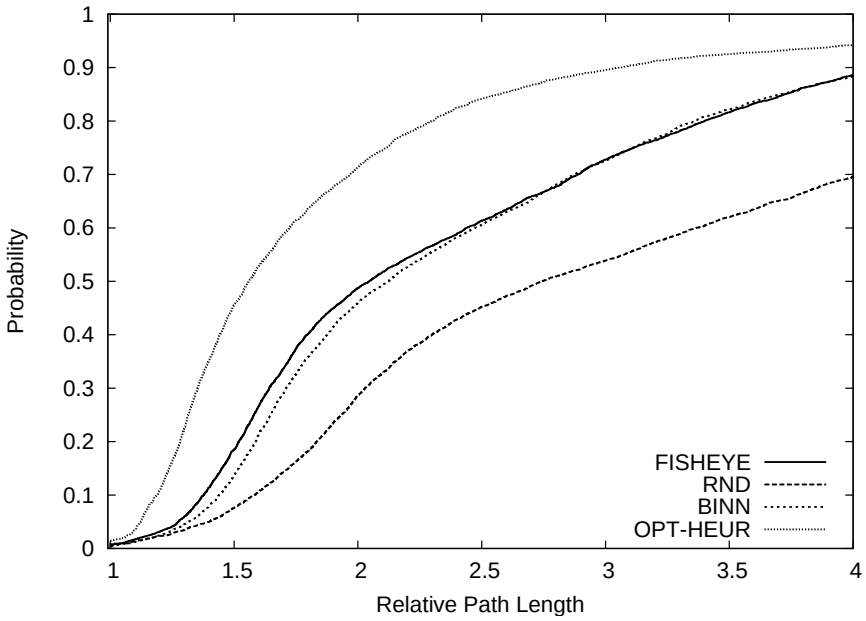

Figure 5.5: CDF of relative path lengths in the overlay network with 7 neighbors for the King data set.

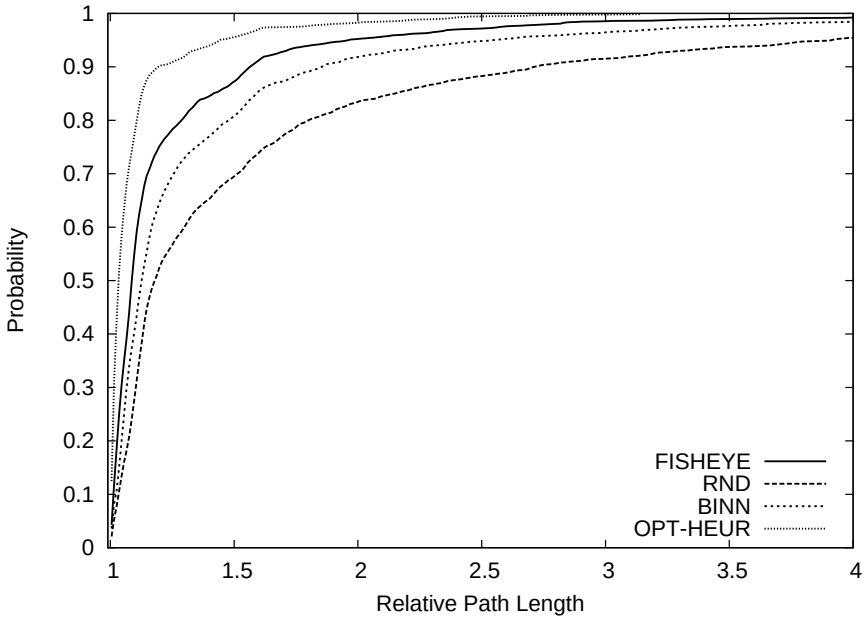

Figure 5.6: CDF of relative path lengths in the overlay network with 21 neighbors for the Planet Lab data set.

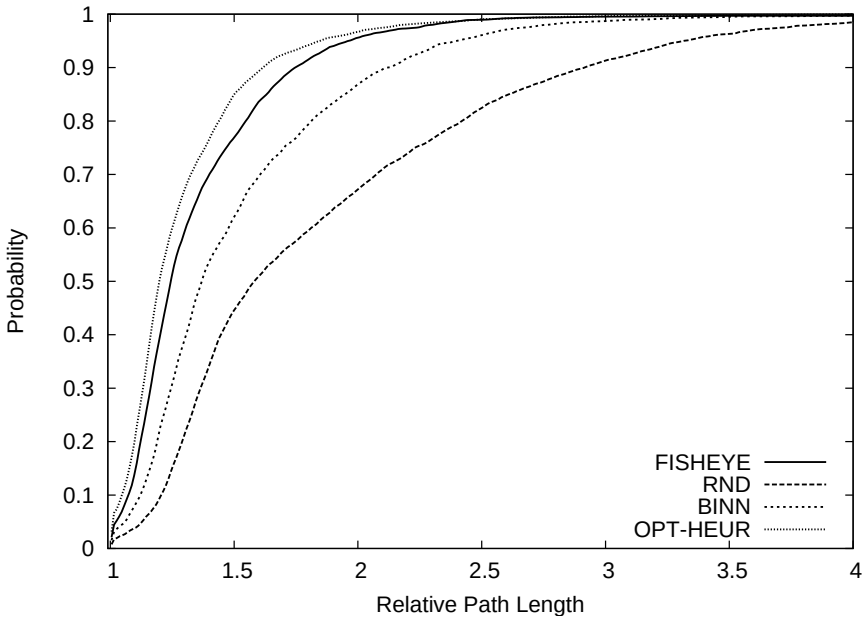

Figure 5.7: CDF of relative path lengths in the overlay network with 21 neighbors for the King data set.

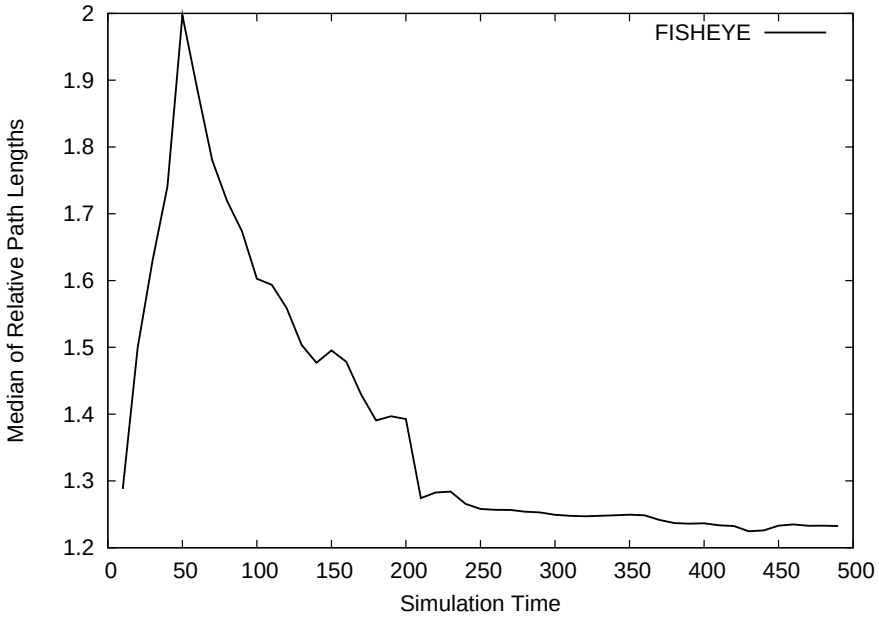

Figure 5.8: Convergence at high churn of the FISHEYE overlay network with 19 neighbors using the King data set.

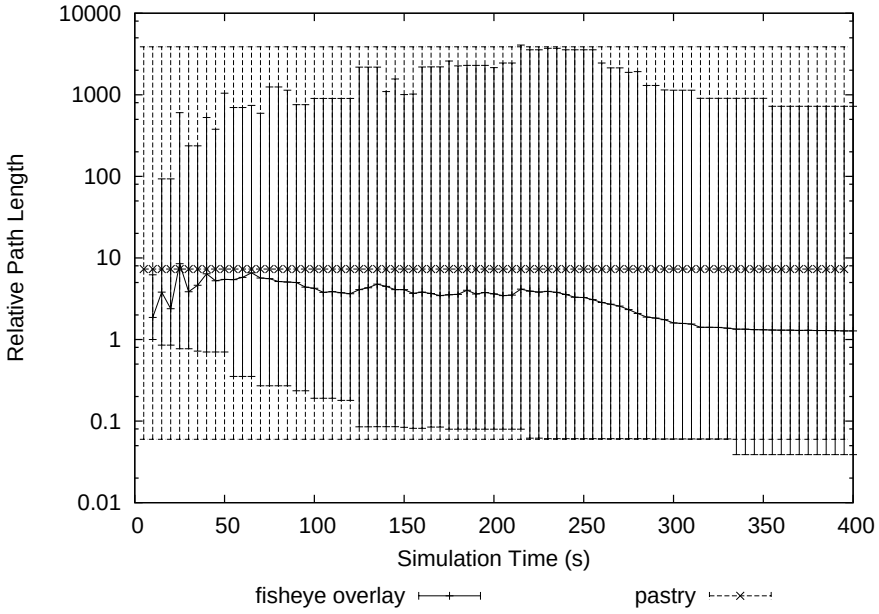

Figure 5.9: Relative path lengths of FISHEYE overlay network and Pastry

Chapter 6

NetICE9: A Stable Landmark-Less Network Positioning System

6.1 Introduction

In the previous chapter, we introduced a protocol for construction of topology-aware overlay networks: fisheye overlay network. In this chapter we use this overlay network in order to improve VIVALDI [16] embedding in virtual spaces.

The main idea behind an RTT prediction scheme is not to measure the RTT between each pair of hosts in the network, but to obtain more or less precise estimates of RTTs between all hosts in the network based on a small amount of measured RTTs. Such an RTT prediction scheme can be used to optimize overlay network structures, to choose the nearest data source for file transfers, or to find an optimal route based on predefined QoS parameters.

As described in chapter 2.1 there are two approaches to embeddings end system into virtual metric spaces: landmark-less and landmark-based. In this chapter, we propose NetICE9, an improvement of the landmark-less VIVALDI [16] approach.[1]. NetICE9, like VIVALDI, is a distributed simulation of a physical system. This means that each host calculates its position in the virtual space without knowing the whole system. The only information that each host has is about its neighbors. Unlike VIVALDI, which simulates a system of hosts connected by springs, NetICE9 is a simulation of a crystallization process. This means that NetICE9 simulates the creation of a crystal structure in a virtual space, where each host corresponds to one atom within this structure. The forces that determine the relative positions of the atoms are proportional to the difference between measured RTTs and distances in the virtual space representing them. In a crystal, those forces are in balance. Essentially, each atom is positioned in such a way that the forces of repulsion from and attraction to its neighbors are in balance.

In this chapter, we also present results of our simulations based on RTTs measured in the Internet [41, 42]. Comparing the performance of NetICE9 and VIVALDI with different

[1]The name NetICE9 was inspired by a fictional ice isomere named ice-nine described in Kurt Vonnegut's novel *Cat's Cradle*

data sets, our simulations clearly show that our approach (NetICE9) yields significantly better results in terms of both the stability of host positions and the precision of the RTT embedding.

This chapter is structured as follows: In the following section, we present the most significant RTT embedding schemes and other related work. In Section 6.2, we identify the sources of instability of VIVALDI. Section 6.3 introduces NetICE9, an improved adaptation of VIVALDI. In Section 6.4 we present our evaluation methodology and results of our simulations. Section 6.5 summarizes the results of the research presented in this chapter.

6.2 VIVALDI's Instability

VIVALDI has numerous advantages over other RTT embedding schemes. Being fully distributed, it does not require any infrastructure and is very robust against churn. It also has pretty good RTT prediction properties, scales well with the number of hosts participating and is simple to implement. On the other hand, as we show in this Section, VIVALDI tends to be unstable. By instability we mean permanent change of host positions in the virtual space. This permanent change of host positions in the virtual space has two sources. One source are the local oscillations of the host positions relative to its neighbors. These oscillations are relatively small and result in a small change of RTT predictions obtained through VIVALDI.

The more severe source of the instability is the movement of the whole system, i.e. all hosts participating in the distributed simulation performed by VIVALDI. Since VIVALDI never reaches a stable state, local oscillations never cease. Those small local oscillations of one hosts affects the hosts with it as a neighbor. This change then affects their neighbors and so on. As the final effect, the whole system does not stand still in the virtual space, but instead it translates and rotates.

Movement of the whole system in the virtual space does not change the RTT prediction obtained through VIVALDI much, since the relative positions of the hosts in the virtual space do not change much. But, if we consider how such positions are used, it turns out that the system movement poses a problem. Positions of the hosts are usually used in order to predict RTTs. If the positions are constantly changing, we must obtain a new position of the host, to which we would like to have an RTT estimate. By doing so, we could also just perform an RTT measurement itself, since it involves the same amount of communication. Hence, being able to rely on a host position for a longer time period is desired.

6.2.1 How Bad is the Instability?

As an illustration of the problem we performed the following experiment: We implemented VIVALDI in an event based simulator [60]. For the network model, we used a RTT distance matrix obtained through "all sites ping" experiment of Planet-Lab presented in chapter 2.4.1. This RTT distance matrix contains RTTs for each pair of hosts participating in Planet-Lab [40]. The total number of hosts in this data is 217. Each host obtained a random choice of 18 neighbors, to which it performed RTT measurements and optimized its position in a

virtual 5-dimensional Euclidean space. The goal of our experiment was to determine, for how long one could rely on positions of other hosts.

To do so, we obtained the position of one host 2 seconds after the start of the simulation. We kept this position constant and used it to determine the relative embedding error to all other hosts in the system defined as

$$\left| \frac{\text{measured rtt} - \text{predicted rtt}}{\text{measured rtt}} \right|$$

at different time points in the simulation. Figure 6.1 shows the result of the experiment. The whole system tends to move away from the fixed point at more or less constant speed. This motivated us to define a measure of the instability of VIVALDI (and related

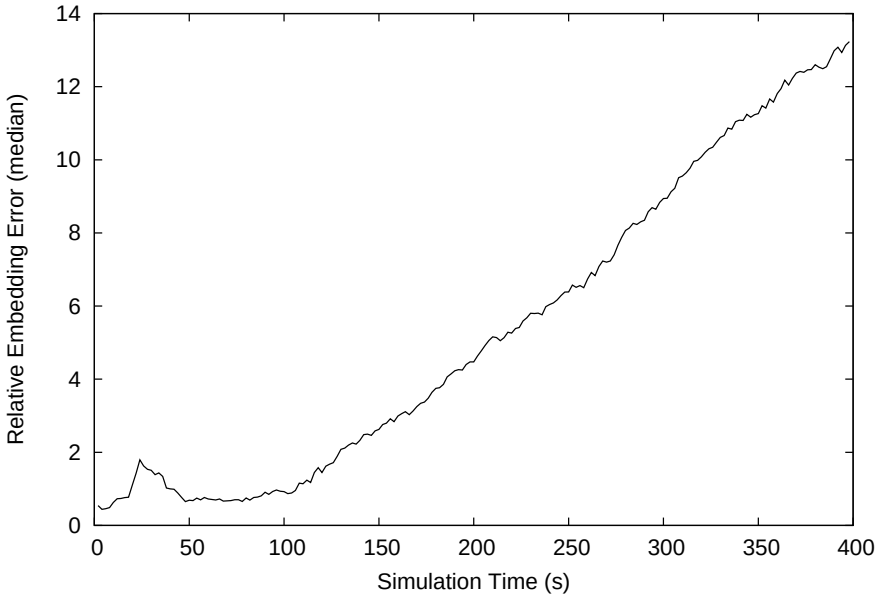

Figure 6.1: Average relative embedding error of the fixed position of one host relative to changing positions of the whole system

approaches): average host speed. Each host "travels" through the virtual space by updating its position. At each position update, there is a distance the host has made, defined as the distance between the old and the new position of the host. We define the speed of a host to be the sum of those distances (calculated from the beginning of the simulation) divided by the simulation time. This measure gives us an estimate of how unstable one system is: the bigger the average speed of the hosts in the virtual space is, the more unstable is the system.

In order to test the correctness of our VIVALDI implementation, we compared the behavior of the same simulation with two different RTT data sets. One data set is the same we used to obtain results from Figure 6.1 (VIVALDI Planet-Lab). The second one is a RTT matrix we obtained by randomly placing 217 points in a 5 dimensional Euclidean space and

calculating distances between them (VIVALDI metric). If our implementation of VIVALDI is correct, the embedding error of the VIVALDI approach using the VIVALDI metric data, which stems from the Euclidean space, should reach 0 after some convergence time. At the same time, the average host speed should also reach 0.

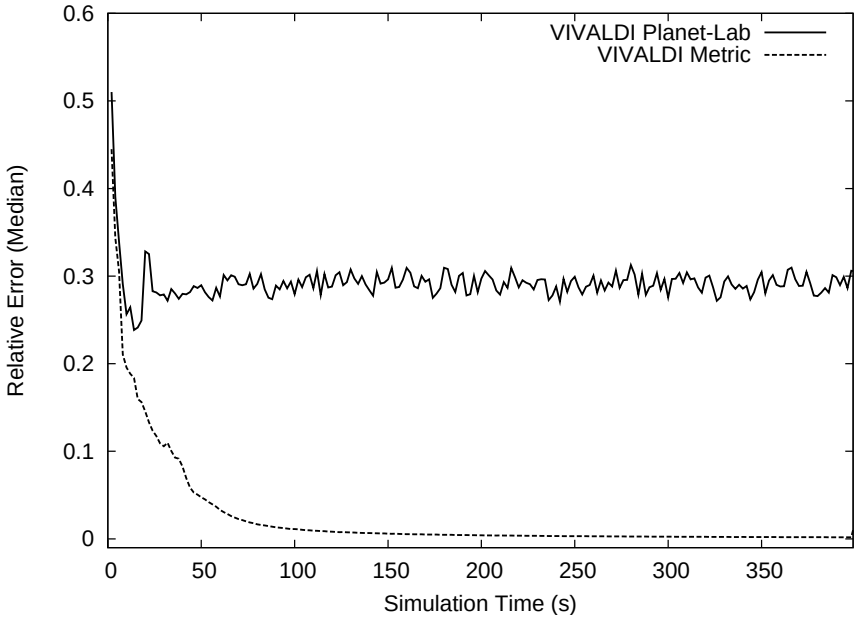

Figure 6.2: Median of the relative embedding error using RTTs measured in Planet-Lab vs. distances originating from a metric space

Figure 6.2 shows the median of the embedding error for the two data sets. As we can clearly see, the median of the relative embedding error of the simulation using the VIVALDI Metric data converges after 150 s of the simulation towards 0. At the same time, the relative embedding error of VIVALDI Planet-Lab remains more or less constant.

In Figure 6.3 we show the average host speed for both data sets. The average host speed of VIVALDI Metric reaches 0 at the same time the relative embedding error reaches 0. On the other hand, the average host speed of VIVALDI Planet-Lab converges towards a constant value, meaning that the whole system is constantly moving through the virtual space.

6.2.2 Reasons for VIVALDI's Instability

Three hosts whose RTTs do not satisfy the triangle inequality may be sufficient to render a VIVALDI system unstable. For example, those RTTs could be $2.5, 3$ and 8. Each time one host tries to correct its position relative to another host, it will either decrease the "predicted" value of the RTT 8 or increase one of the other two RTTs (2.5 or 3) to have larger predicted values. Since VIVALDI always optimizes one (randomly chosen) RTT prediction, it never

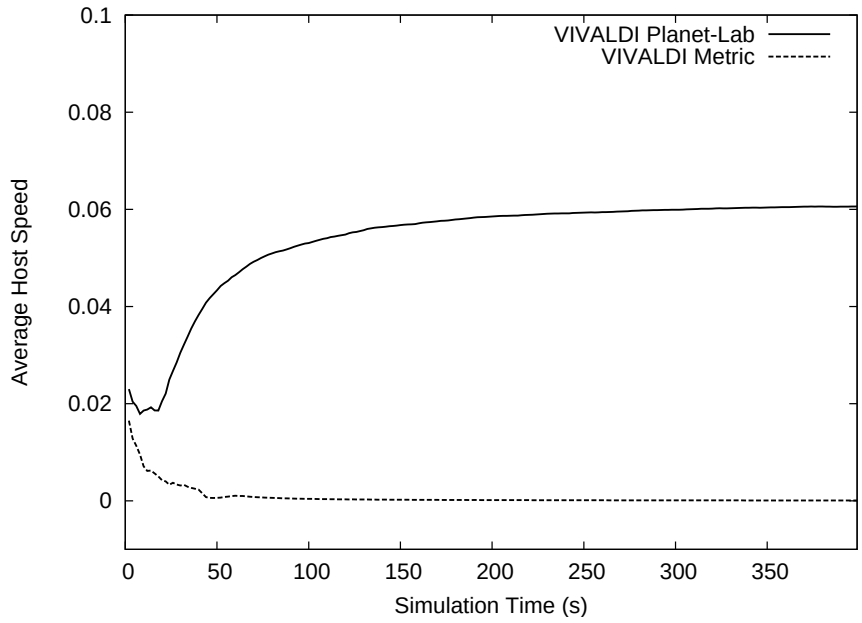

Figure 6.3: Average (moving) speed of host using RTTs measured in Planet-Lab vs. distances originating from a metric space

reaches a stable state, since decreasing the error in one direction increases the error in the other, which must be then compensated in turn. Figure 6.4 illustrates such a behaviour, that we describe as "host chasing". At each simulation step, one hosts tries to reduce its RTT prediction to an other host. By doing so, it increases the embedding error towards all other hosts. This increased embedding error then gets corrected by another host, which in turn increases other embedding errors. The result of this behavior is the translation of the whole system through the virtual space.

Another drawback of VIVALDI arises from the fact that VIVALDI does not require the "is a neighbor" relation to be bidirectional. A bidirectional "is a neighbor" relation means, that if host A optimizes its position relative to host B, then B should optimize its position relative to A. In practice, each host chooses a fixed number of random neighbor hosts and optimizes its position in the VIVALDI system relative to them. This means, if there is no optimal solution (i.e. some of the properties of the metric space are violated by the measured RTTs), hosts will be oscillating around their (theoretically) optimal position.

Figure 6.5 show what could happen, when the "is a neighbor of" relation is only one way. The host in the middle of the figure tries optimizing its position relative to the neighbors, which at each optimization step leads to a new position of the host, since the neighbors will never move themselves towards the host in the center. If the relation "is a neighbor of" would be bi-directional, the neighbors of the host in the center would also move towards that host, which would reduce oscillation of its position.

6.3 NetICE9

6.3.1 Crystallization

The basic idea of NetICE9 is to simulate the process of crystallization. Within NetICE9, each host is considered to be an atom that should be embedded into a crystal structure. The crystal structure consists of other atoms positioned within the virtual space in such a way that the forces of repulsion and attraction between them are in balance. We define the force between two atoms to be the difference between the predicted and measured RTT. If this difference is negative, there is a repulsion force between two atoms. Positive difference means the attraction force between two atoms.

Each host determines its own position within the crystal structure. In order to do so, it has a set of neighbors. To each of those neighbors, it periodically measures the RTT. At the same time, the host also queries each of the neighbors about its current position in the virtual space and its current moving speed (how fast did the neighbor change its position in the virtual space lately). The host uses this information in order to update its own position. Updating the position is done by minimizing the objective function (6.1). This objective function is very similar to the one used by GNP, with the difference that we are using neighbors as landmarks and that each neighbor is weighted according to its moving speed. The larger the moving speed of the neighbor is, the lower is the weight. The whole process of updating the position of a host is described in Algorithm 6.1.

The rationale behind using such an algorithm is determining the position of the host using all available information. VIVALDI uses a much simpler approach, where the optimization is done only towards one neighbor. In the case, where there is no optimal embedding possible, this approach will result in oscillations of the host positions. Those oscillations then have a rippling effect on the whole system, as shown in Section 6.2. If we perform the optimization in the way we propose in Algorithm 6.1, the host position will remain stable, if the positions, velocities and RTTs remain stable. We assume that this would greatly improve both stability and precision (reduce the embedding error) of the system.

$$f_e(\mathcal{C}_\mathcal{H}) := \sum_{i=1}^{m} \frac{1}{1 + (\mathcal{V}_{\mathcal{N}_i} \cdot 1000)} \cdot (d(\mathcal{C}_\mathcal{H}, \mathcal{C}_{\mathcal{N}_i}) - \hat{d}_{\mathcal{H}\mathcal{N}_i})^2 \tag{6.1}$$

6.3.2 Choice of Neighbors

A crystal structure implies aligning of atoms relative to its direct neighbors. Doing so in order to obtain an RTT embedding is not the best idea. The authors of VIVALDI [16] showed that using only nearest neighbors for embedding yields in bad results of predicting RTTs of distant neighbors. Instead, they propose using a small portion of distant neighbors, in order to give a host a sense of global position in the network.

We were also assuming that using a bidirectional “is a neighbor of” relation would significantly reduce the rippling effect of local oscillations of hosts to the whole system. The reason for this assumption is that if a bidirectional “is a neighbor of” relation is used,

Algorithm 6.1 NetICE9 algorithm for updating host position in a virtual space

Require: N Set of neighbors
Require: T_u Update interval
Require: P_c Current position of the host
Require: V_p Previous EWMA of host's speed
Require: α Dampening factor for EWMA
  for $n \in N$ **do**
    $(RTTS[n], POS[n], SPEED[n]) \leftarrow Query(n)$ {Query a neighbor about its current speed (EWMA), position and measure RTT to it}
  end for
  $P_p \leftarrow P_c$ {Store the current position of the host as the previous position}
  $P_c \leftarrow Minimization(RTT, POS, SPEED, P_p)$ {Determine the new position of the host using function minimization. Parameters for the objective functions are RTTs, current positions and current speed of the neighbors. Use the previous position of the host as the starting point for the function minimization.}
  $delta \leftarrow |P_p - P_c]$ {Calculate the distance between the old and the new position}
  $V \leftarrow delta/T_u$ {Calculate the moving speed of the host}
  $V_p \leftarrow \alpha \cdot V + (1 - \alpha) \cdot V_p$ {Update the EWMA of the host velocity}
  return P_c, V_p

the rippling effect of local oscillations will be reduced, since the effect of local oscillations of two hosts would go both ways and cancel out each other. The problem was that there are not many neighbor selection strategies that have both of those properties. For this reason we developed own neighbor selection strategy we described in chapter 5.

A fisheye-view of an overlay network is a choice of c hosts from the overlay network with special properties. Those properties are geographical diversity and fisheye distribution of neighbors. Geographical diversity means that the choice of neighbors is evenly distributed around the host. Fisheye distribution of neighbors means that the density of chosen neighbors decreases with the distance to the host.

We achieve those properties by performing a distributed gravity force minimization algorithm as described in chapter 5. An interesting feature of our fisheye overlay approach is, that it does not require embedding of hosts into a virtual space. Instead, it is based only on measured RTTs.

We also developed a version of our overlay network building protocol, which is able to create a fisheye overlay network with bidirectional "is a neighbor" relation. This overlay network is the one we are using as a neighbor selection for our NetICE9 approach.

6.4 Evaluation

To evaluate the precision and stability of our approach, we compared it to the precision and stability of VIVALDI. We also compared its precision to that of GNP. To do this, we implemented VIVALDI in an event-based network simulator [60].

6.4.1 Simulation Scenario and Parameters

The network model of the simulation is the same one used by Dabek et al. in their original evaluation of VIVALDI [16]. Each message within the network is delayed by half the RTT between two hosts. The RTT information for our model was obtained from RTT measurements from the Internet. For input data we hat two data sets. One set (denoted as Planet-Lab) contains a full RTT distance matrix of 217 different hosts obtained from the "all sites ping" experiment (see chapter 2.4.1). The other data set (denoted as KING) contains a full RTT distance matrix of 462 hosts, which was obtained using the King method (for details see chapter 2.4.2).

In each run of the simulation, we started a new instance of a host every 100ms until all hosts were active. This means that all hosts are active by the time the simulation has been running for 21.7 s in the case of the Planet-Lab data or after 46.2 s in the case of the KING data set. For each embedding we used a 5-dimensional Euclidean space as the virtual space. We limited the number of chosen neighbors (c) to maximally 18. We kept each simulation running for 800 s (simulation time).

6.4.2 Impact of Neighbor Selection on VIVALDI

One of the first things we noticed in our simulations was that selecting neighbors unidirectionally at random may not be the best choice. In order to confirm this, we performed a simulation of VIVALDI with three different neighbor selection strategies.

- *RAND* selects c neighbors at random and uses them for the optimization.This strategy is more or less the one used by VIVALDI.
- *FISHEYE-UD* uses the fisheye in a unidirectional mode. With this strategy, each host chooses a fisheye view of the overlay network. The relation "is a neighbor of" is not guaranteed to be bidirectional in this strategy.
- *FISHEYE-BD* is a fisheye in a bidirectional mode. This strategy differs from the unidirectional fisheye in that the resulting overlay network is guaranteed to be two-way (bidirectional "is a neighbor of" relation).

Figures 6.6 - 6.9 show the average embedding error and average host speed for all three proposed neighbor selection strategies for both data sets.

The results of the simulations show that, as we expected, using a better overlay network can result in more stable host positions. Figures 6.7 and 6.8 show an interesting result: Using a fisheye overlay without bidirectional "is a neighbor of" relation actually results in an even more unstable system. This is because the fisheye method of host selection increases the effect where hosts are "chased" through the virtual space. For that reason, the prediction error of FISHEYE-UD is even worse than that of RAND. As soon as the choice of the neighbors is bidirectional, the "chasing"-effect disappears and the systems becomes both more stable and more precise compared to a random choice of neighbors (RAND).

6.4.3 NetICE9 vs. VIVALDI

The second improvement we made to VIVALDI is to position a host relative to all known neighbors simultaneously instead of optimizing the position relative to one neighbor at a time. For this purpose, as in the previous comparison, we compared the median of the relative embedding errors and average host speed in the virtual space for VIVALDI using a random neighbor choice, VIVALDI with a bidirectional fisheye neighbor choice and NetICE9 (a combination of bidirectional fisheye neighbor choice and positioning relative to all neighbors). We performed this comparison for both the Planet-Lab and the KING data set. Results of this comparison can be seen in Figures 6.10 - 6.13.

6.4.4 Prediction Error

Since the median of the relative prediction error just gives an estimate of the average prediction error, we decided to take a closer look at the overall distribution of the relative RTT embedding error. In order to achieve this, we compared CDFs (cumulative distribution functions) of predicted distances. In Figures 6.14 and 6.15, we present the CDFs of the relative embedding error for NetICE9, GNP, VIVALDI RAND and VIVALDI FISHEYE-BD. Those CDFs have been obtained by comparing relative embedding errors of each of those systems after running the simulation for 400 seconds. In order to have comparable GNP results we also used embedding into a 5 dimensional virtual space using 18 landmarks. To reduce the impact of randomness to the result, we compared the result of 30 simulation runs, each of them made with a different initial seed for the pseudo random number generator used by the simulator. This seed both influences the choice of neighbors, landmarks and initial starting points for function minimization.

Both Figures 6.14 and 6.15 show that NetICE9 clearly outperforms VIVALDI in terms of relative error of RTT embedding. Also, a somewhat interesting result is that NetICE9 outperforms the embedding of GNP. We explain this by the fact that NetICE9, due to its distributed nature, uses a larger portion of available RTT measurements. This enables NetICE9 to obtain a better embedding compared to GNP, since GNP uses only RTTs to landmarks, which are all the same for all other hosts. Another surprising result can be seen in Figure 6.15, which shows that using the bidirectional fisheye overlay network improves the performance of VIVALDI almost to the level of the performance of GNP.

6.5 Conclusion

In this chapter we have identified two major sources of VIVALDI's instability: First, it does not choose its neighbors symmetrically. Second, it optimizes towards only one neighbor at a time and disregards all information known about the other neighbors. In order to avoid these problems, we proposed a new approach we named NetICE9. Similar to VIVALDI, NetICE9 is also a fully distributed simulation of a physical system. Unlike VIVALDI, however, NetICE9 simulates the creation of a crystal structure. In this crystal structure, every atom (i.e. every host) positions itself relative to its surrounding atoms (hosts).

In order to have a stable system, we chose the surrounding of each atom in such a way that the whole crystal remained stable (i.e. the atoms of the crystal do not move). To achieve this, we propose using such a choice of neighbors so that the "is a neighbor of" relation is bidirectional (A is a neighbor of B iff. B is a neighbor of A). Also, motivated by results presented by the authors of VIVALDI, the choice of neighbors should be geographically diverse. This means that for an ideal RTT prediction scheme, a mix of close and remote neighbors should be used. One choice of neighbors fulfilling both of those properties is the fisheye [54] overlay network.

We have evaluated the NetICE9 approach by comparing it to VIVALDI and GNP. For the evaluation we used a simulation based on RTTs measured in the Internet. Our evaluation focused mainly on aspects of stability and precision of RTT embedding. The evaluation showed that NetICE9 outperforms both VIVALDI and GNP in the terms of precision of RTT embedding. At the same time, the NetICE9 approach proved to be more stable than VIVALDI (as long as RTTs do not change, GNP has no instability per se).

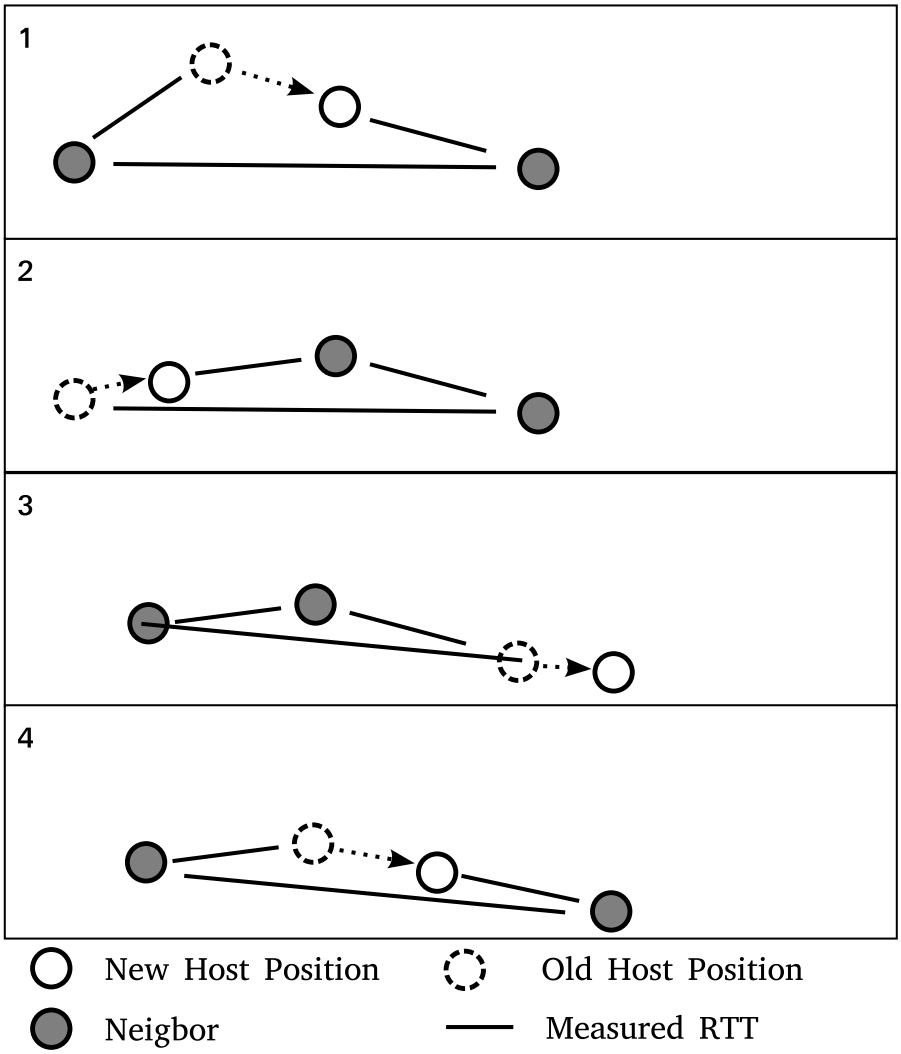

Figure 6.4: An example of "host chasing" in VIVALDI caused by a triangle inequality violation

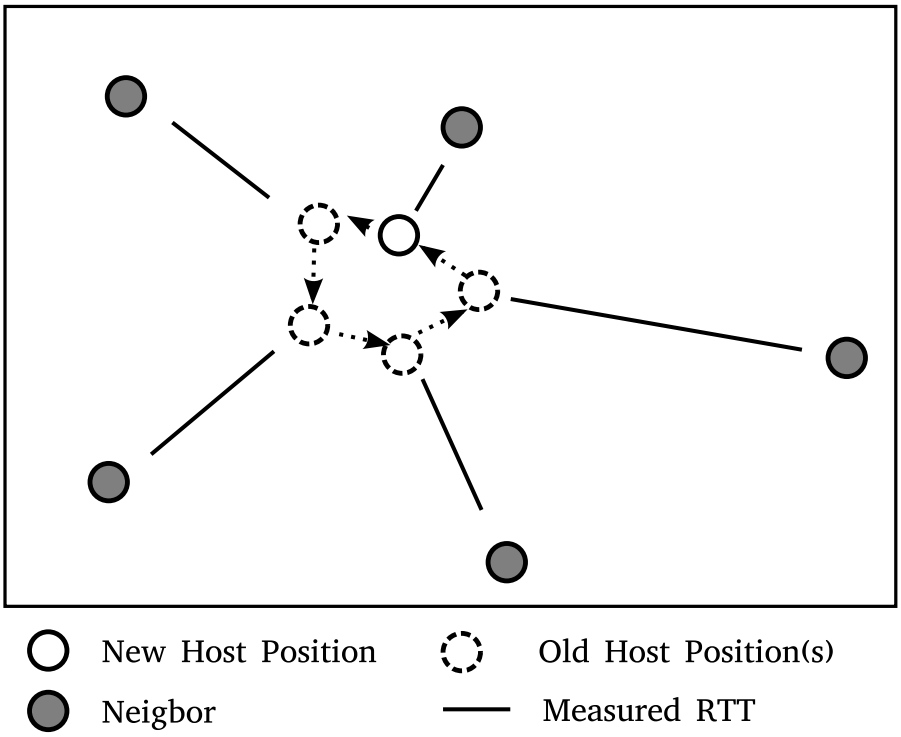

Figure 6.5: An example of oscillations in VIVALDI caused by lack of bi-directional "is a neighbor of" relation

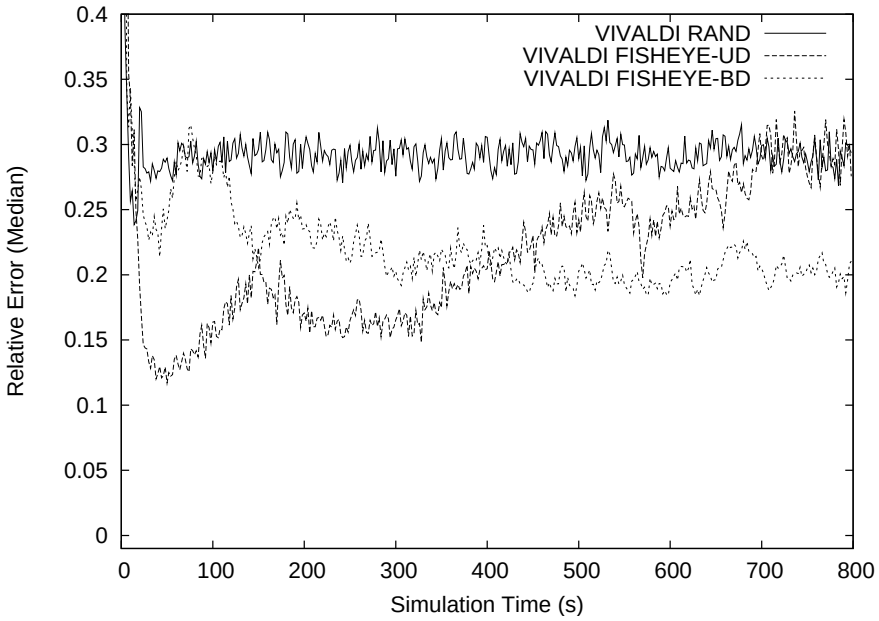

Figure 6.6: Median of embedding error for VIVALDI using different neighbor selection strategies with Planet-Lab data.

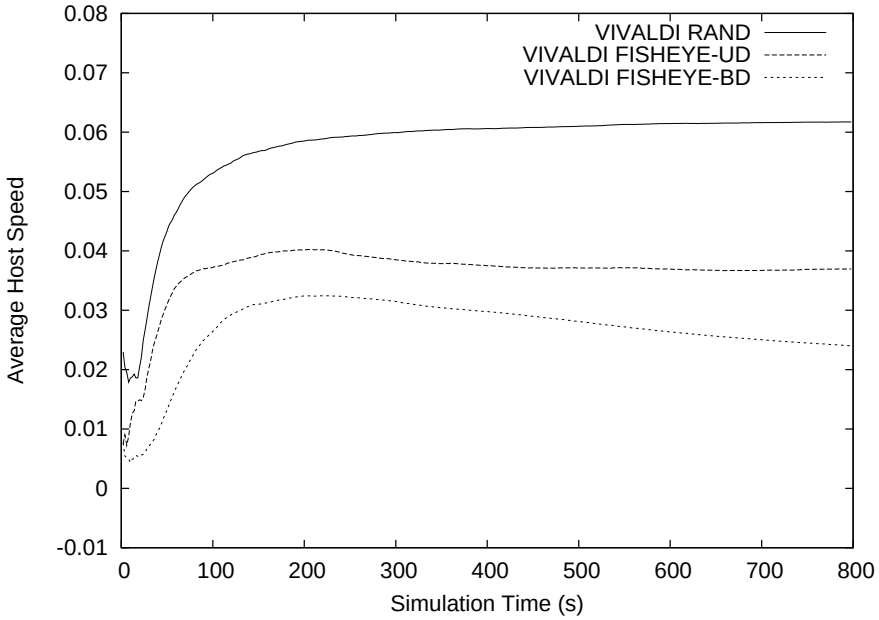

Figure 6.7: Average host speed in the virtual space for VIVALDI using different neighbor selection strategies with Planet-Lab data.

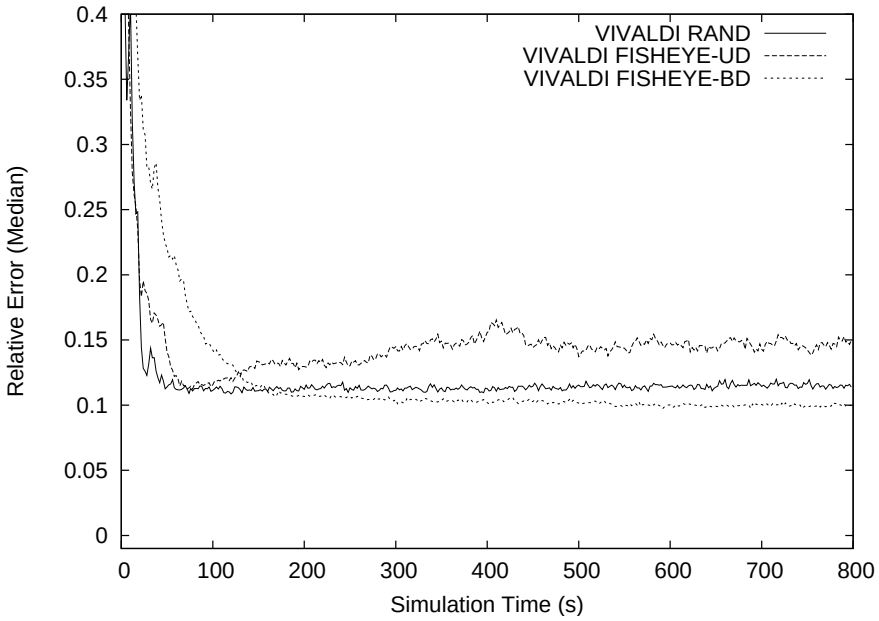

Figure 6.8: Median of embedding error for VIVALDI using different neighbor selection strategies using KING data.

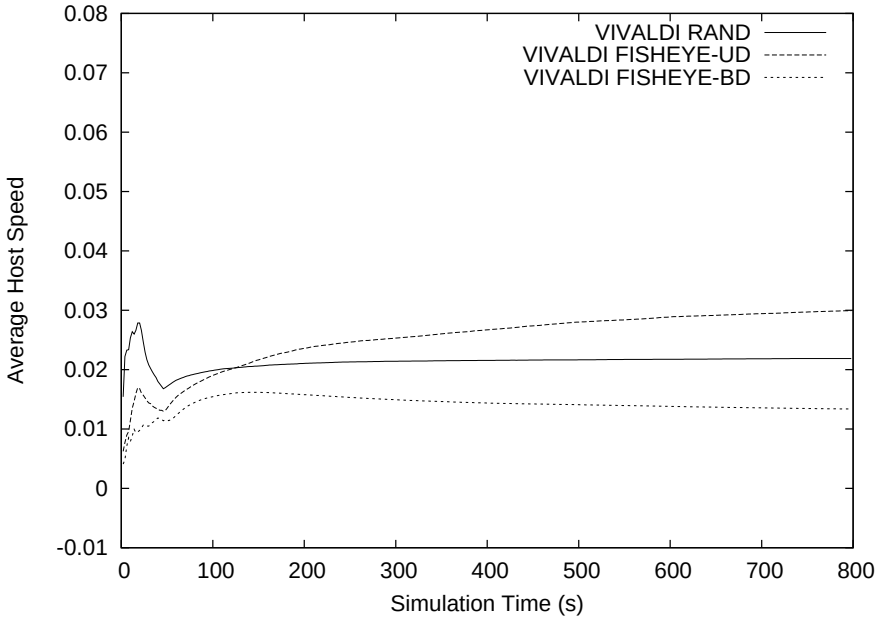

Figure 6.9: Average host speed in the virtual space for VIVALDI using different neighbor selection strategies with Planet-Lab data.

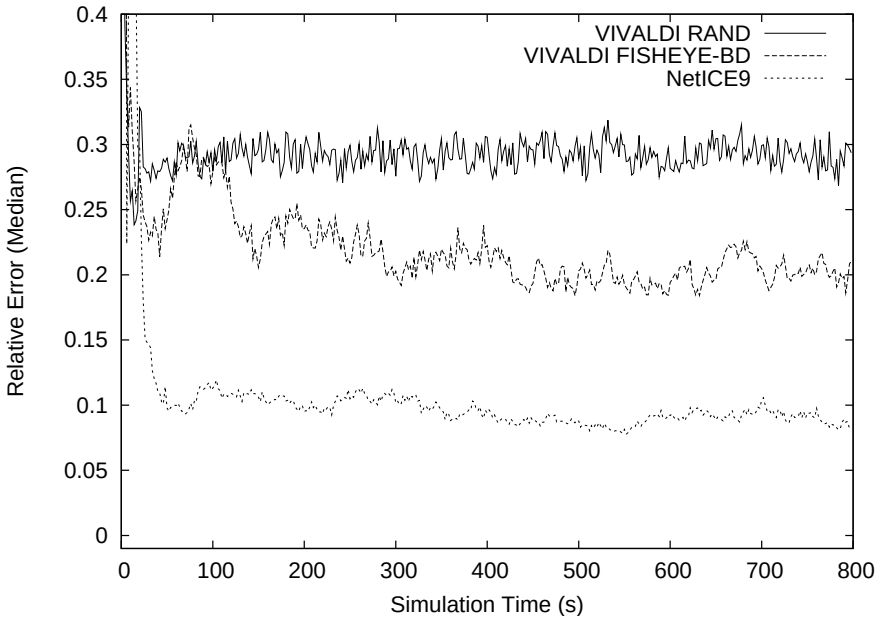

Figure 6.10: Median of embedding error for NetICE9 compared with VIVALDI using Planet-Lab data.

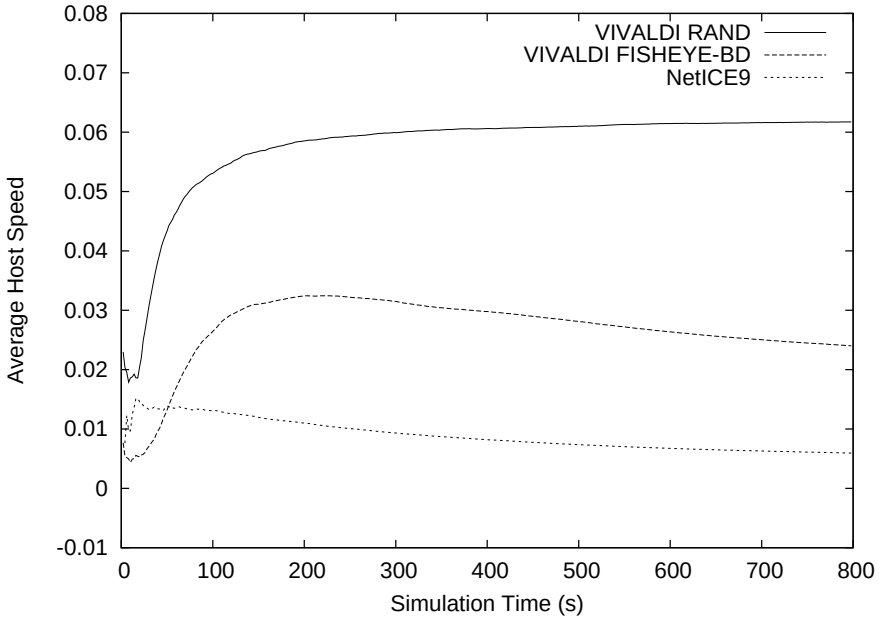

Figure 6.11: Average host speed in the virtual space for for NetICE9 compared with VIVALDI using Planet-Lab data.

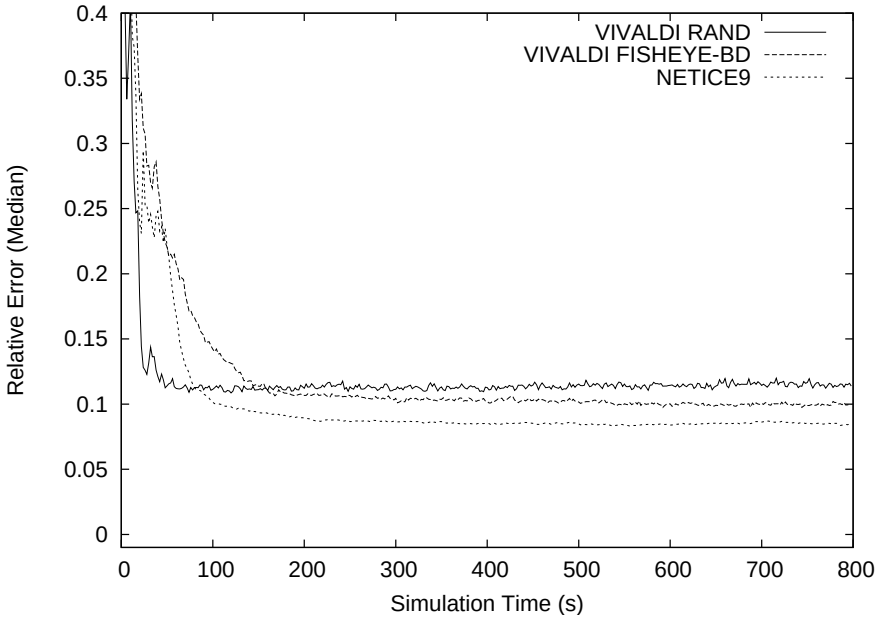

Figure 6.12: Median of embedding error for NetICE9 compared with VIVALDI using KING data.

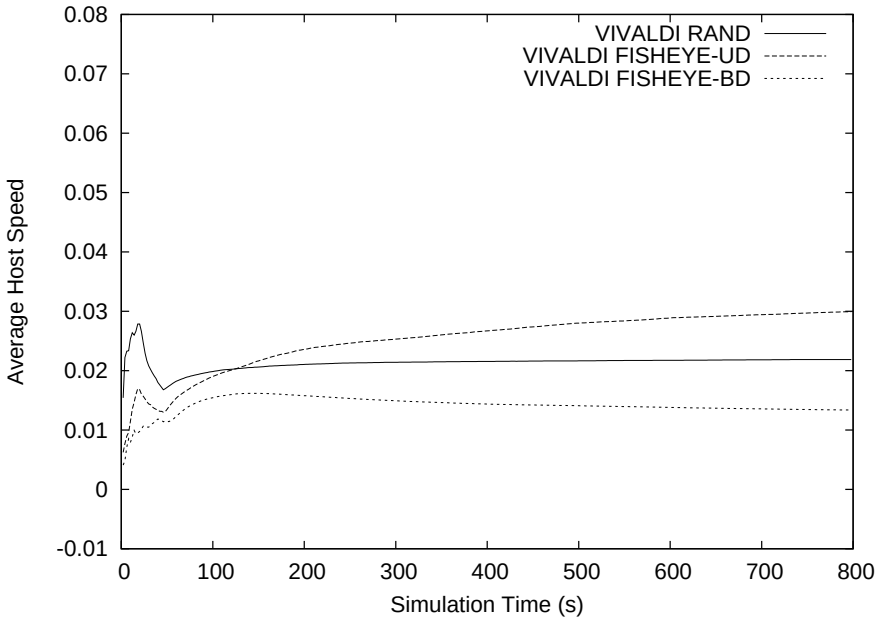

Figure 6.13: Average host speed in the virtual space for for NetICE9 compared with VIVALDI using KING data.

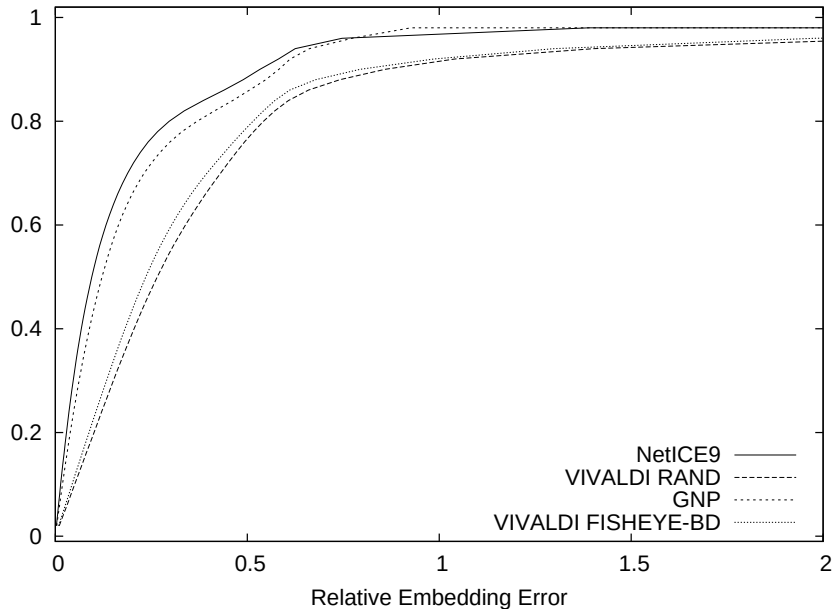

Figure 6.14: Comparison of embedding error CDFs using Planet-Lab data.

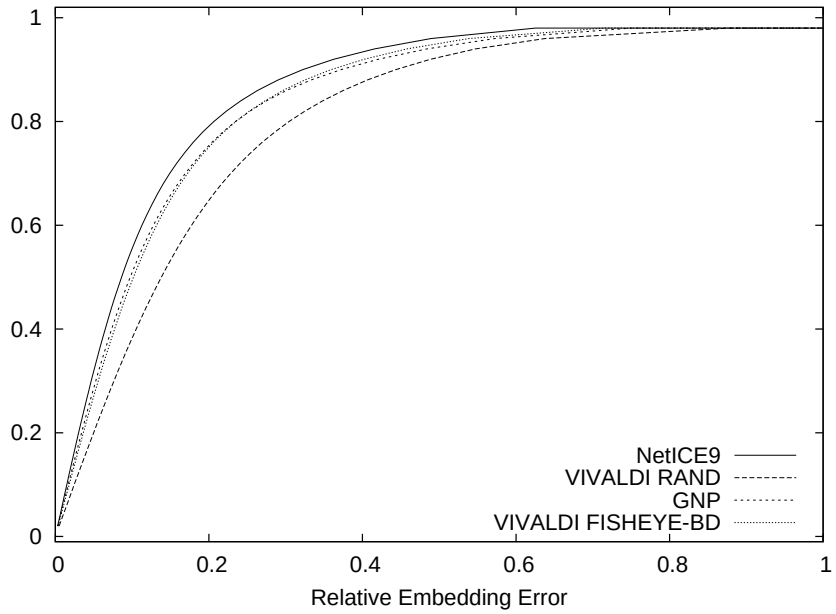

Figure 6.15: Comparison of embedding error CDFs using KING data.

Chapter 7

Practical Applications of Round Trip Time Embedding

7.1 Introduction

If we consider an overlay network to be a weighted graph, as proposed in chapter 1.2, then finding a route between two end systems in an overlay network can be reduced to the problem of finding a cycle-free path between the two vertices representing those end systems in the graph of the overlay network. We can quantitatively compare different paths between the same two end systems by comparing their sum of weights (RTT stretch). The lower this sum, the better is the path. Since the number of vertices and edges in such a graph is finite, the number of possible cycle-free paths between two end systems is finite also. If we consider all possible paths, there is at least one path with a minimal RTT stretch. This path we call the optimal path.

Unfortunately, in order to find the optimal path in the graph, we need to know the whole graph. This poses a problem since every end system in the overlay network has only a limited amount of memory. If every end system had to store the information of the complete overlay network, the maximal size of the overlay network would be limited by the available memory on each end system. Thus, in order to have a scalable overlay network, we must accept a trade-off between the optimality of found paths and memory requirements of the routing protocol. Ideally, the amount of memory required on each end system is constant and independent of the size of the overlay network.

Most of the currently existing structured overlay routing protocols [6, 8] require a constant amount of memory on each end system, but they totally disregard the RTT stretch of the found paths. Instead, they optimize the number of hops (number of end systems on the path). As a result, they find routing paths with few hops, but with a high RTT stretch. Different suggestions have been made to fix these approaches so the RTT stretch of the found paths is reduced and thus the approach is made more topology-aware [7, 34]. Still, the resulting paths are far from being optimal.

In this chapter, we present an approach to routing in an overlay network which takes advantage of the fact that we know the positions of the end systems in a virtual space. As

presented so far in this dissertation, the embedding of end systems into a metric space is constructed to be used as an RTT prediction scheme. In this chapter, we outline a new way of using such embeddings – geographic routing. The idea behind our approach is to exploit the fact that in the process of performing the embedding, information concerning the topology of the underlying network is extracted. Since RTT information can be used to model the structure of a communication network, embedding end systems into a virtual space where the distances between end systems represent the RTTs between them is a kind of black box approach to topology extraction. This topology extraction can be used to provide other topology-aware services.

In this chapter, we present applications of such an embedding to provide topology-aware unicast and multicast routing. We also present a way to provide locally bound flooding services which can be used for service discovery.

The structure of this chapter is the following. In the next Section, we introduce greedy routing, which is the most popular geographic routing approach. In Section 7.3, we present the problems which can be encountered when applying greedy and other geographic routing protocols in overlay networks. Section 7.4 introduces the nearest neighbors convex set (NNCS) overlay network. NNCS overlay networks are easy to build and to maintain in a distributed manner and, furthermore, in them greedy routing is always successful. We use this overlay network not only for unicast routing: In Section 7.6, we present a multicast service based on flooding on reverse greedy unicast paths to the servers in our overlay network.

7.2 Greedy Routing

7.2.1 The Principles of Greedy Routing

As previously mentioned, our goal is to use greedy routing as this allows us to keep routing decisions simple. Greedy routing compares the distances of each known neighbor to the destination. The neighbor that is nearest to the destination is chosen as the next hop, i.e. the message is sent to that neighbor. This procedure is continued until the message reaches its destination.

More formally explained, the procedure of greedy routing is the following: When an end system R with neighbors $\{N_1 \dots N_c\}$ whose positions in the virtual space are $\{\mathcal{C}_{N_1} \dots \mathcal{C}_{N_c}\}$ receives a message for destination D with position $\mathcal{C}_D$, it calculates the distances between its neighbors and the destination: $\{d_n | d_n := d(\mathcal{C}_{N_n}, \mathcal{C}_D)\}$. The neighbor N_m with d_m, the minimal distance to the destination, is chosen for the next hop.

What makes greedy routing problematic is the requirement that at each routing decision we are able to choose a neighbor which is closer to the destination. If we are unable to find such a neighbor, our routing protocol can end up in a loop and is never able to deliver the message at its destination. If we were able to build an overlay network that guarantees that at each end system there is at least one neighbor which is closer to the destination, independently of the position of the destination, than the current end system, greedy routing would always be successful.

Forwarding messages through an overlay network is done in the following way: The sender of a message includes the position of the receiving end system in the message. Every end system that receives a message for which it is not the final recipient relays the message to the neighbor that is nearest to the destination. Figure 7.1 illustrates an example of such a message delivery path.

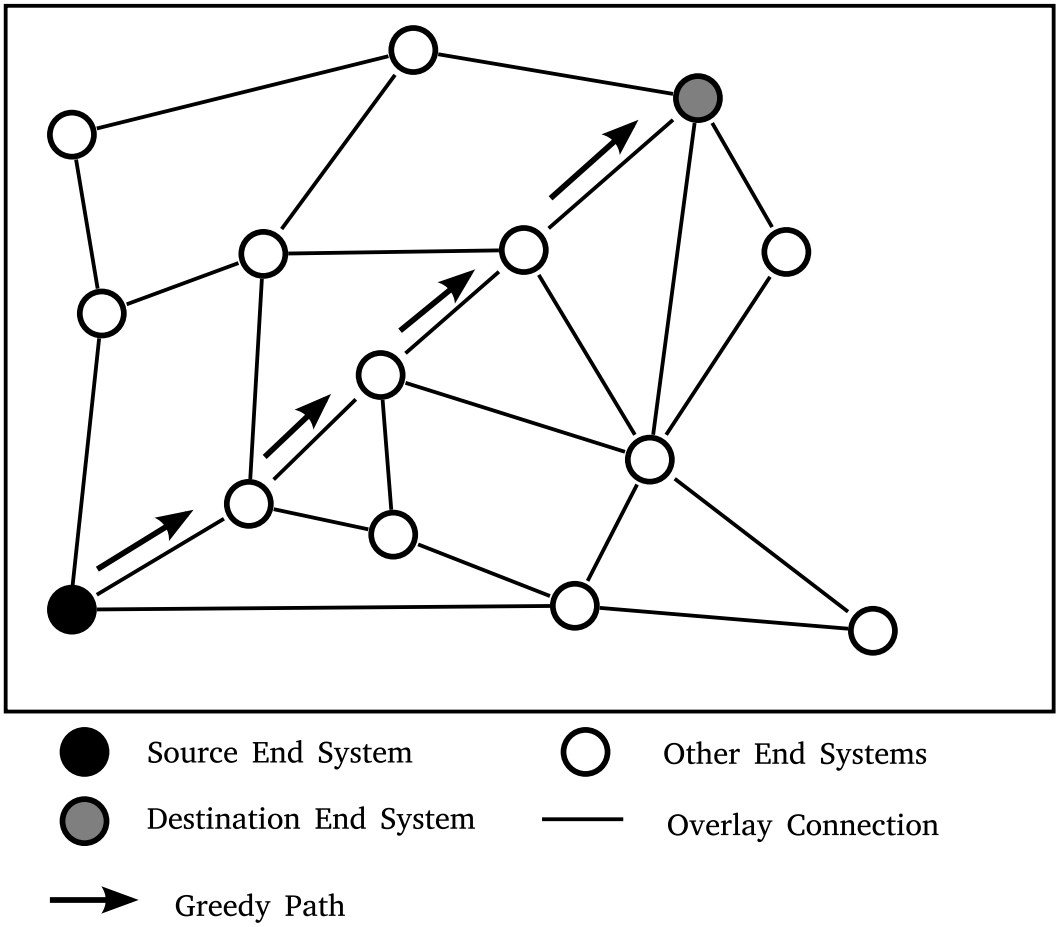

Figure 7.1: Example of Greedy Routing in a Virtual Space

Unfortunately, if we use just any overlay network such as [54] or [34], there is no guarantee that we will be able to make "progress" toward the destination (for details see the next Section). This means that in the Fisheye overlay network, we will not be able to use the greedy routing protocol. In Section 7.4, we will show that if end systems have positions in a metric space, we can construct an overlay network in which the success of greedy routing is guaranteed.

7.2.2 Applications of Greedy Routing

Using geographical routing protocols such as greedy routing is not a new idea. Greedy routing has been mainly used in mobile ad-hoc networks (MANETs) where every mobile node is aware of its physical position (usually GPS coordinates) [62].

The greedy routing used in MANETs differs in many respects from the greedy routing proposed in this dissertation. For example, greedy routing in MANETs requires only two or at most three dimensions, because it uses positions in the physical space. Furthermore, in a MANET, each end system's radio transceiver has a limited range of transmission, which, in most cases, restricts its range of communication to direct neighbors. This is not the case in general overlay networks. Here, we assume that each end system can communicate directly with every other end system via the underlying network.

Another significant difference is the mobility of end systems. In MANETs, every end system can change its position at any time. In overlay networks, end systems change their position in the virtual space only if there has been a significant change in the underlying network. Still, the basic premises remain the same: Each end system knows its own position as well as that of the destination end system, it has limited memory and limited available bandwidth.

7.3 Problems of Greedy Routing

The main problem of greedy routing in any kind of network is posed by non-convex "holes" in the network. If the shape outlined by end systems and edges around a hole in the graph is non-convex, we refer to it as a non-convex hole. The presence of non-convex holes leads to the failure of greedy routing. One example of this problem is illustrated in Figure 7.2. In this example, greedy routing decisions lead to a loop where the message is sent back and forth between two end systems. The message never reaches its destination.

To solve this problem, greedy routing schemes in regular overlay networks have a backup-mode which is used when greedy routing fails.[1] For example, the Right-Hand rule in GPRS is such a backup mode. Here, when greedy routing fails, a graph is traversed using the Right-Hand rule until greedy rooting can be used again. However, the Right-Hand rule can only be used in two-dimensional graphs. In general Euclidean graphs, especially with dimensions higher than two, the Right-Hand rule cannot be applied.

Fortunately, it is possible to construct an overlay network in such a way that the creation of non-convex holes is prevented. If we can guarantee that each end system is able to choose a relay for the message which is closer to the destination, greedy routing will be always effective.

[1] For clarity, in this thesis, overlay networks which cannot guarantee the success of greedy routing are referred to as regular overlay networks.

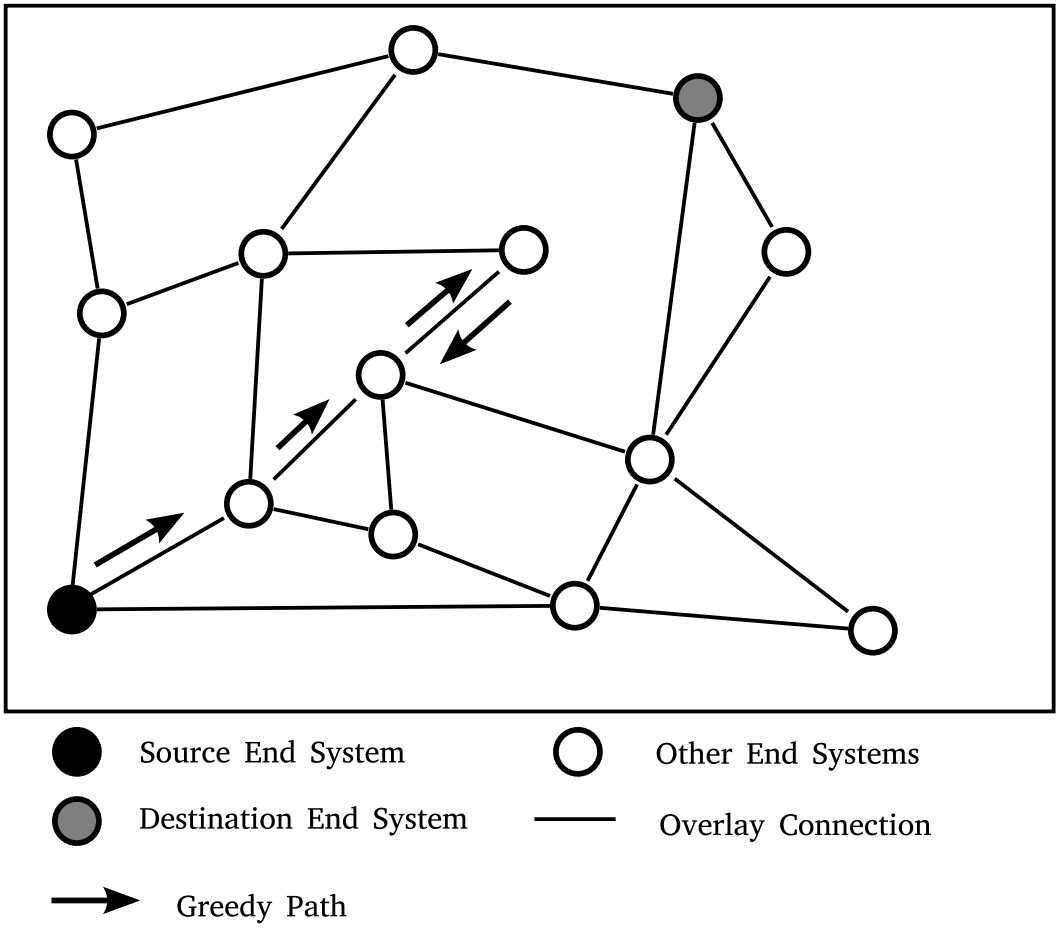

Figure 7.2: Example Greedy Routing Failure Due to a Non-Convex Network Hole

7.4 Nearest Neighbors Convex Set (NNCS) Overlay Network

7.4.1 Requirements for Overlay Networks

Unicast routing in an overlay network is equal to finding a cycle-free path between two vertices which correspond to end systems in the directed weighted graph representing the overlay network. Ideally, the process of finding this path can be limited to making one decision at each end system. Each time an end system receives a unicast message that is not addressed to it, it has to forward it to one of its neighbors. The choice to which neighbor the message should be forwarded is known as the routing decision. In this Section, we assume that every end system is part of an overlay network and is embedded into a k-dimensional Euclidean space. In our approach, we will use the end systems' position information to help us make a routing decision on each end system.

We also assume that, due to its participation in the overlay network, each end system

knows the location and address within the underlying network of its direct neighbors. This information is provided by the overlay network protocol implementation. One possibility for such an overlay network is the one proposed in chapter 6. We only require that the overlay network protocol provides each end system with the knowledge of its direct neighbors and makes sure that this information is kept up to date. Furthermore, the overlay network protocol has to ensure that the overlay network is a connected graph. Else, no routing protocol can ever find a path to every destination within the overlay network.

7.4.2 Nearest Neighbors Convex Set (NNCS)

In order for greedy routing to succeed, we must make sure that on each "hop" within the overlay network we are able to progress toward the destination end system, i.e. to reduce the remaining distance to it. There are different methods to achieve this. One of them is to create a Delaunay triangulation of the overlay network. In such a triangulation, greedy routing is always successful [63].

The problem is that constructing a Delaunay triangulation of an overlay network is non-trivial. This is due to the fact that while building a Delaunay triangulation in a distributed manner is possible, it is a task in which end systems need to cooperate and share states. The communication protocol and the algorithm required to achieve this are potentially complicated. We propose a far simpler approach involving the nearest neighbors convex set.

There is only one constellation of factors in which greedy routing can become "stuck". This happens when a message reaches an end system which is a local maximum in respect to the progress towards the destination position. An end system is a local maximum in respect to the progress towards the destination if it is closer to the destination than any of its direct neighbors. If we can guarantee that for any possibly destination position within the virtual space each end system has at least one neighbor that guarantees progress towards the destination, then greedy routing is always successful. We ensure that this property is fulfilled by requiring that each end system knows its nearest neighbors convex set.

We define the nearest neighbors convex set (NNCS) as follows: For any given end system R with position $\mathcal{C}_R$ in the virtual space, its NNCS contains c end systems $\{H_1, \ldots, H_c\}$ with positions $\{\mathcal{C}_{H_1}, \ldots, \mathcal{C}_{H_c}\}$ which are neighbors of R. NNCS is the convex set defined by the intersection of the half-spaces S_j, $j \in \{1, \ldots, c\}$. Each of the half-spaces S_j is bound by the hyperplane which is orthogonal to the line containing both $\mathcal{C}_R$ and $\mathcal{C}_{H_j}$ and contains $\mathcal{C}_R$, as depicted in Figure 7.3. The nearest neighbors convex set has the following properties:

- The NNCS of R contains R and its direct neighbors $\{H_1, \ldots, H_c\}$.
- Every other end system H_x that is not a neighbor of R ($H_x \notin \{H_1, \ldots, H_c, R\}$), is outside NNCS of R.

Figure 7.4 depicts an example of a NNCS in two dimensions.

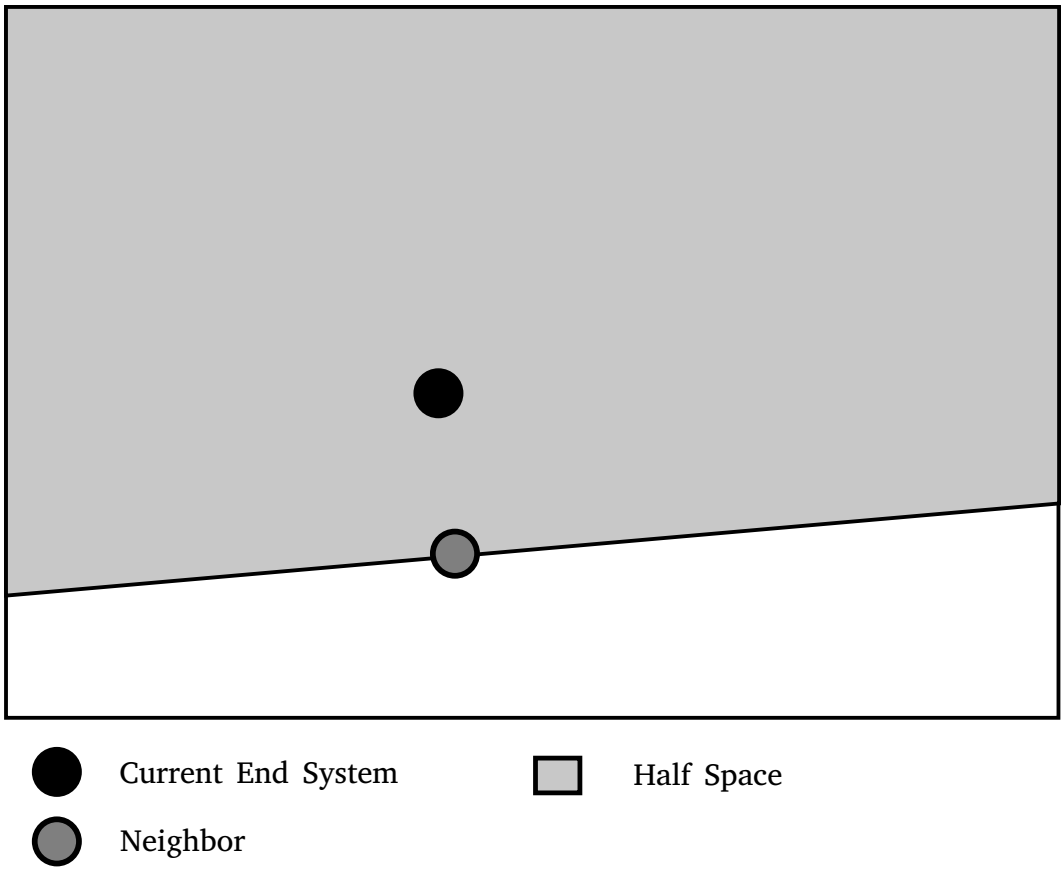

Figure 7.3: Half-Space Defined By One Neighbor

7.5 Greedy Routing in an NNCS

One of the main advantages of an NNCS is that it guarantees that greedy routing makes progress, i.e. that it reduces the distance to the destination at each routing hop. The mathematical proof for this statement is outlined in the following.

We prove our argument by contraposition: We assume that in the NNCS, there exists at least one end system serving as a routing hop at which greedy routing fails, i.e. it is not able to make more progress towards the destination. We designate this end system as R. R receives a message M_D which is destined for end system D. At this stage, greedy routing chooses the neighbor H_n as the next hop. We now assume that the distance $d(\mathcal{C}_{H_n}, \mathcal{C}_D)$ is greater than or equal to the distance $d(\mathcal{C}_R, \mathcal{C}_D)$. In other words, we assume that greedy routing is not successful, i.e. that the message does not make progress toward its destination.

Now, if $d(\mathcal{C}_{H_n}, \mathcal{C}_D) > d(\mathcal{C}_R, \mathcal{C}_D)$, then the half space defined by the two points $\mathcal{C}_R$ and $\mathcal{C}_{H_n}$ will inevitably contain both R and D. In fact, there will be no half space in the NNCS of R that does not also contain D. If there were a half space that contained R but not D,

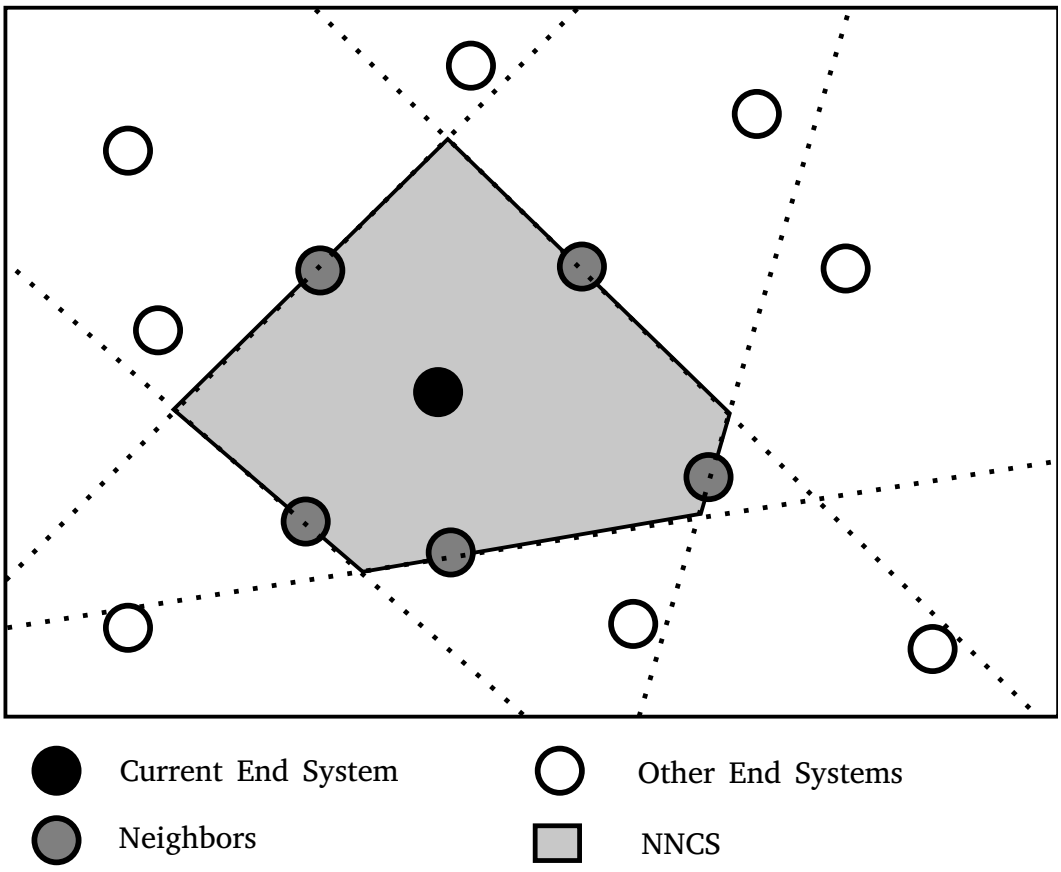

Figure 7.4: Example of a Nearest Neighbors Convex Set (NNCS) in Two Dimensions

then the neighbor H_x defining that space would be closer to D than to R, i.e. we would have found a neighbour H_x for which $d(\mathcal{C}_{H_x}, \mathcal{C}_D) < d(\mathcal{C}_R, \mathcal{C}_D)$, which in turn would contradict our initial assumption.

Thus, we have shown that all half spaces of the NNCS of $\mathcal{C}_R$ contain $\mathcal{C}_D$. In other words, D lies within the NNCS of R. This means that D is one of the direct neighbors of R. This contradicts the assumption that greedy algorithm would choose another neighbor, since $d(\mathcal{C}_D, \mathcal{C}_D) := 0$. Hence, our assumption is not correct, meaning that every "hop" in the overlay network reduces distance to the destination.

Since the number of end systems in the overlay network is finite and at every routing hop the distance to the destination is reduced, greedy routing is always successful in an NNCS overlay network.

7.5.1 Building an NNCS Overlay Network

It is virtually impossible for an individual end system to store a global view of the overlay network in which it participates. For this reason, each end system needs to have a method of discovering and maintaining its *NNCS*. Since each end system R has its own *NNCS*, which is independent of the *NNCS*s of other end systems, NNCS overlay networks can be maintained relatively easily.

As in any other overlay network, each end system joining the network must know at least one end system already participating in the overlay network. In other words, the new end system N_n knows at least one end system B, which is already a part of the *NNCS* overlay network. Since N_n knows end system B, it puts B in its *NNCS*. In a next step, N_n queries B about its *NNCS*. After receiving the information of the *NNCS* of B, N_n uses Algorithm (7.1) to update its own *NNCS*. This procedure is repeated until the *NNCS* of N_n is stable, i.e. until no new neighbors are accepted into the new *NNCS*. Once the *NNCS* of N_n has stabilized, N_n periodically queries its neighbors about their *NNCS*s. If one of the neighbors does not respond to repeated queries, N_n assumes that that particular neighbor has left the overlay network and removes it from its *NNCS*.

Algorithm 7.1 Algorithm for maintaining the NNCS of an end system

Require: R_{NNCS}: NNCS set of R
  for $N \in R_{\text{NNCS}}$ **do**
    $N_{\text{NNCS}} \leftarrow query\_neighbor(N)$ {Query neighbor to get its current NNCS}
    for $N_i \in N_{\text{NNCS}}$ **do**
      if N_i is inside convex set defined by R_{NNCS} **then**
        $R_{\text{NNCS}} \leftarrow R_{\text{NNCS}} \cup \{N_i\}$ {add N_i to R_{NNCS}}
      end if
      $R_{\text{NNCS}} \leftarrow minimize\_nncs(R_{\text{NNCS}})$ {Remove all neighbors from R_{NNCS} that are not in the NNCS}
    end for
  end for
  return R_{NNCS}

Algorithm 7.2 Algorithm for finding the minimal NNCS of an end system ($minimize\_nncs$)

Require: R_{NNCS}: NNCS set of R
Require: $\mathcal{C}_R$: Position of R in the virtual space
  $S \leftarrow construct\_halfspaces(\mathcal{C}_R, R_{\text{NNCS}})$ {Construct all halfspaces of every neighbor}
  for $N \in R_{\text{NNCS}}$ **do**
    $\mathcal{C}_N \leftarrow get\_position(N)$ {Get position of neighbor in the virtual space}
    for $H \in S$ **do**
      if $\mathcal{C}_N \notin H$ **then**
        $R_{\text{NNCS}} \leftarrow R_{\text{NNCS}} \setminus \{N\}$ {remove N from R_{NNCS} if coordinates of N are not contained in one of the half-spaces}
      end if
    end for
  end for
  return R_{NNCS}

7.5.2 Making NNCS Overlay More Robust

Every overlay network based on end systems has a high churn (end systems joining an overlay network and leaving after a relatively short time of period). Therefore, each end system should know not only its own NNCS but also have the ability to quickly replace neighbors that have left the overlay network. There is also the special case where an end system which is located on the "edge" of the virtual space may end up having only one neighbor in its NNCS; this case is illustrated in Figure 7.5. If this neighbor fails or leaves the overlay network, the end system on the "edge" has no more neighbors. In the worst case, this end system is now isolated from the overlay network. To re-join the network, it would have to go again through the whole bootstrap process.

To avoid such situations, we propose that each end system uses two layers of NNCSs. The first layer consists of the NNCS of the end system. The second layer is the NNCS that is constructed when the neighbors that are in the first layer NNCS are disregarded. Figure 7.6 illustrates this idea. By keeping track of two layers of NNCSs, each end system can quickly replace a neighbor from the first layer that has left the overlay network. At the same time, both layers can be used when choosing neighbors in greedy routing decision.

7.5.3 Optimizing Routing

If routing is based on selecting the next hop from *NNCS* only, then the resulting routing jumps are very short; i.e. each time a routing decision is made, the message is sent to that neighbor within the NNCS which is nearest to the final destination. This behavior may be good enough for routing over short distances. For long distances, however, too many end systems are involved and the delivery of the messages is unnecessarily delayed; each "jump" delays the delivery by a constant amount of time in average.

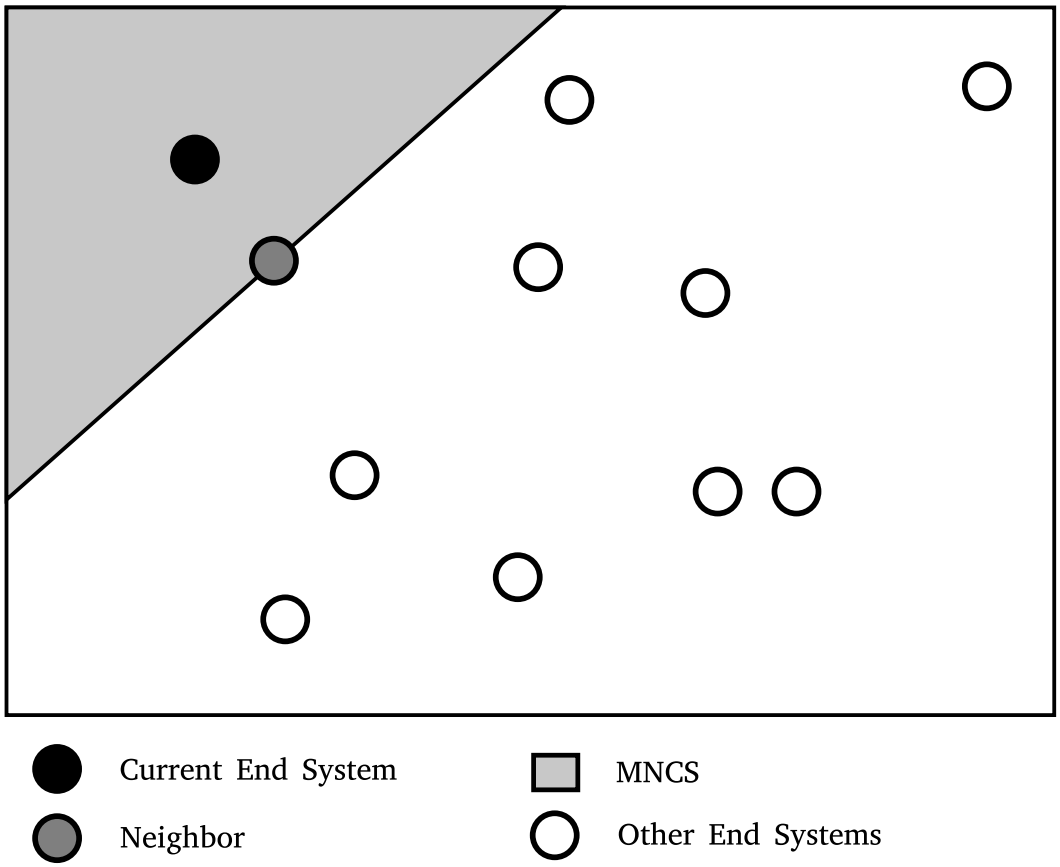

Figure 7.5: Case of an End System Located on the Edge of the Virtual Space with Only One Neighbor

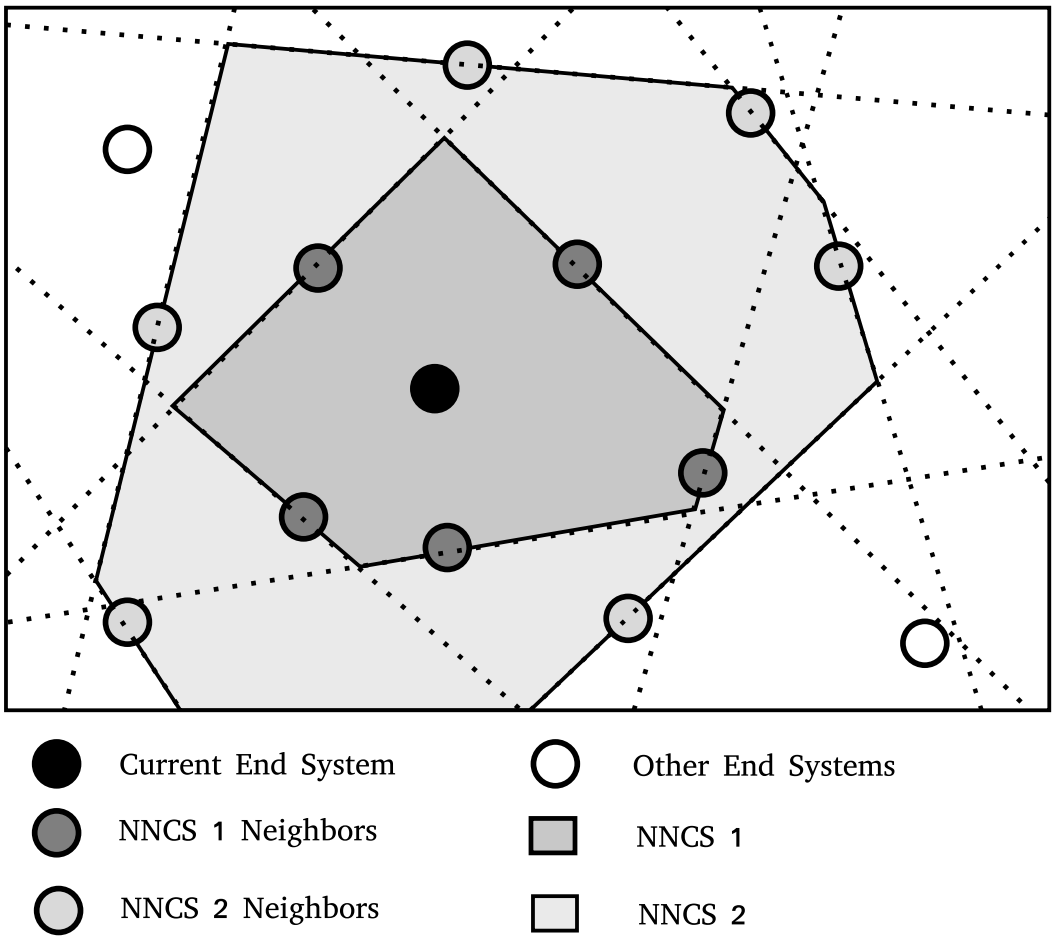

Figure 7.6: The Two Layers of NNCSs for One End System

When relaying a message to a distant destination end system, it would thus be far more desirable if the choice falls on an end system which lies further away from the current end system, and thus closer to the destination. The choice of neigbors offered by *NNCS* is limited to the nearest neighbors, thus, we must widen our selection. In [54], we described the Fisheye overlay network. This overlay network features ideal properties: It offers a relatively tight mesh of close neighbors while still listing some more remote neighbors. However, Fisheye overlay networks do not guarantee the success of greedy routing. The combination of an *NNCS* overlay with a Fisheye overlay results in an overlay network which is able to perform long routing jumps while guaranteeing the success of greedy routing. Hence, for unicast routing we propose that both, a Fisheye overlay and an *NNCS* overlay network be maintained. When a routing decision needs to be made, the greedy algorithm makes its choice from a set of neighbors which contains both *NNCS* and Fisheye overlay neighbors.

7.6 Multicast Routing

Unicast routing using a geographical routing protocol can be useful; however, there are some drawbacks. One of those drawbacks is that the position of the destination within the virtual space must be known. As we have shown in chapter 6, this position can change over the time. Even if we use GNP, the positions of end systems will change as soon as there is a change in the topology of the underlying network.

A more useful application of greedy routing is to use it to construct multicast trees. The idea of multicast routing is the following: The sender (denoted by S) has a position in the virtual space $\mathcal{C}_S$. We assume that every end system participating in the overlay network is interested in receiving multicast data (one overlay network per multicast group). We also require, that every end system participating in the overlay network not only stores and updates its own NNCS, but also keeps track of the NNCSs of all those end systems to which it is a neighbor. This information can easily be collected: Every end system periodically queries its neighbors (as described in Section 7.5.1) and with the query message it can also send its current NNCS.

Every multicast message M_S from the sender S contains the position $\mathcal{C}_S$ of the sender. At each end system, this position is used to determine the recipients of the message. When an end system R receives M_S, it delivers the message locally (since every end system is interested in receiving multicast traffic) and sends a copy of M_S to all those end systems that would use R to route a unicast message towards S. In other words, multicast messages are flooded on the reverse unicast paths towards S. This approach is illustrated in Figure 7.7.

7.7 Service Discovery

An interesting application of topology-aware overlay networks is to use them to find the nearest instance of a specific service. For example, service discovery can be used to find

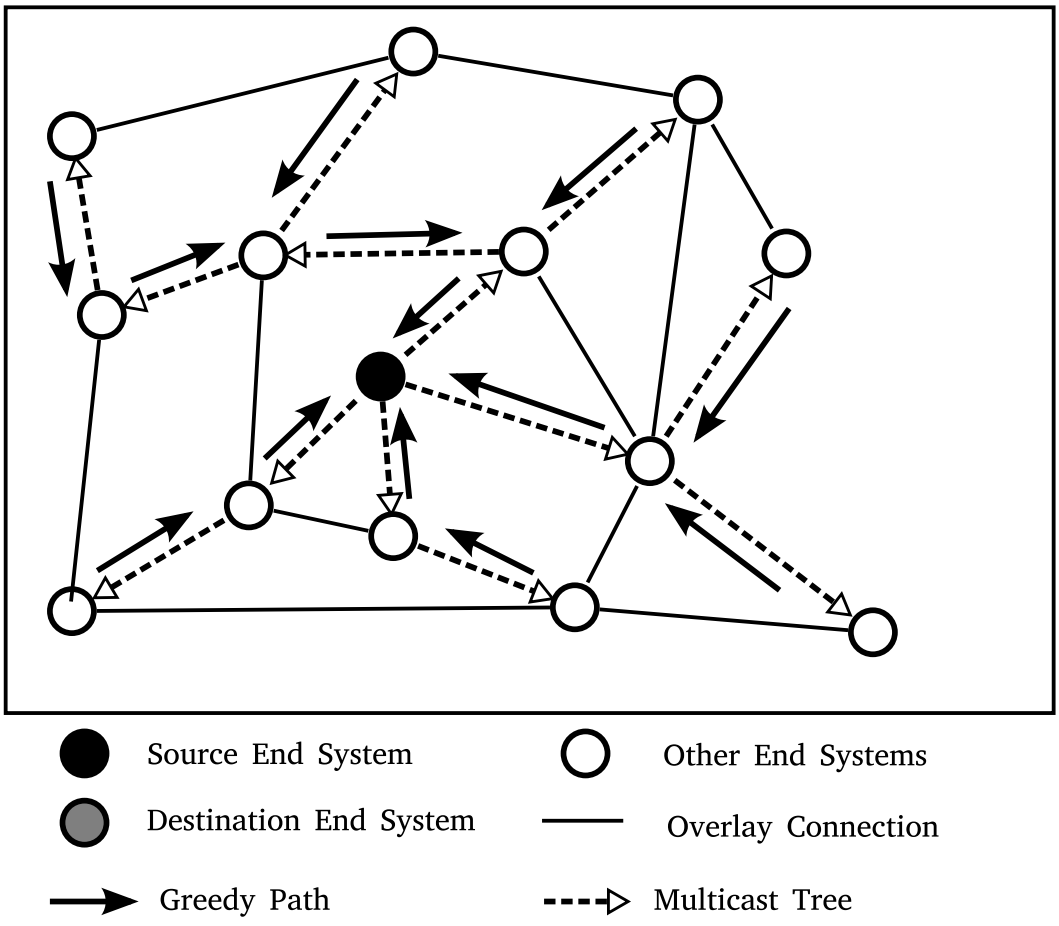

Figure 7.7: Example of Building a Multicast Tree on Reverse Unicast Greedy Routing Paths

the nearest end system that contains a mirror of a file. Service discovery can also help to identify an end system with enough storage or CPU capacity to perform a calculation with special requirements in a cloud computing environment. In both examples, we assume the existence of end systems in the overlay network, which are able to perform such tasks. The real problem is to find them. This task becomes especially difficult if using the service involves the transfer of large amounts of data. In such a case, finding the nearest end system can significantly influence the performance of the whole operation, since a large portion of the transaction would be comprised of data transfer.

Usually, such services are found by flooding the overlay network with request messages. Every end system that matches the query, e.g. has the requested file, can then respond directly to the sender of the message. This is how Gnutella [55], among others, performs its search queries. In Gnutella, every search query floods the overlay network until each query has performed seven hops. Of course, this way of searching an unstructured overlay network is very inefficient, since there is no way to avoid loops, which means that there is a high probability that the same query is delivered to the same end system more than once.

Fortunately, there is a way to provide a similar service in a topology aware manner. For this purpose, we use the algorithm defined in Section 7.6 to flood an NNCS overlay network. In analogy to multicast routing, we flood each end system in the overlay network with query messages along the reverse paths, i.e. along the path that end system would use to reach the sender of the query message.

Flooding begins at end system C. C sends a copy of the query message Q_C to each of the end systems H_n that have C as their NNCS neighbor. In turn, each of those end systems determines which end systems have H_n in their NNCS and would route a message with destination C through H_n. To each of those neighbors, H_n then sends one copy of Q_C. As depicted in Figure 7.8, this procedure is repeated every time an end system receives a query Q_C. This chain of reverse forwarding terminates when none of the end systems that contains H_n in their NNCS would route a message destined for C through H_n.

Furthermore, every end system processes the query Q_C and responds directly to C if it provides the service requested by Q_C. Also, because we are in a metric space, we can limit the radius of the query as represented in Figure 7.9. This excludes those end systems which provide the service but are too far away, i.e. beyond some predefined RTT r. Thus, a query Q_C is forwarded to an end system H only if the position of H ($\mathcal{C}_H$) lies within the hyper sphere around C with radius r. In other words, we require the distance $d(\mathcal{C}_H, \mathcal{C}_C)$ to be smaller than r. By doing so, we reduce the load imposed on the overlay network, since we do not flood the entire overlay network, but only that part of the network that interests us. In the following Section, we expand this idea. We limit the flooding by different shapes within the virtual space in order to provide a service which is otherwise not easily obtainable: The identification of QoS paths within overlay networks.

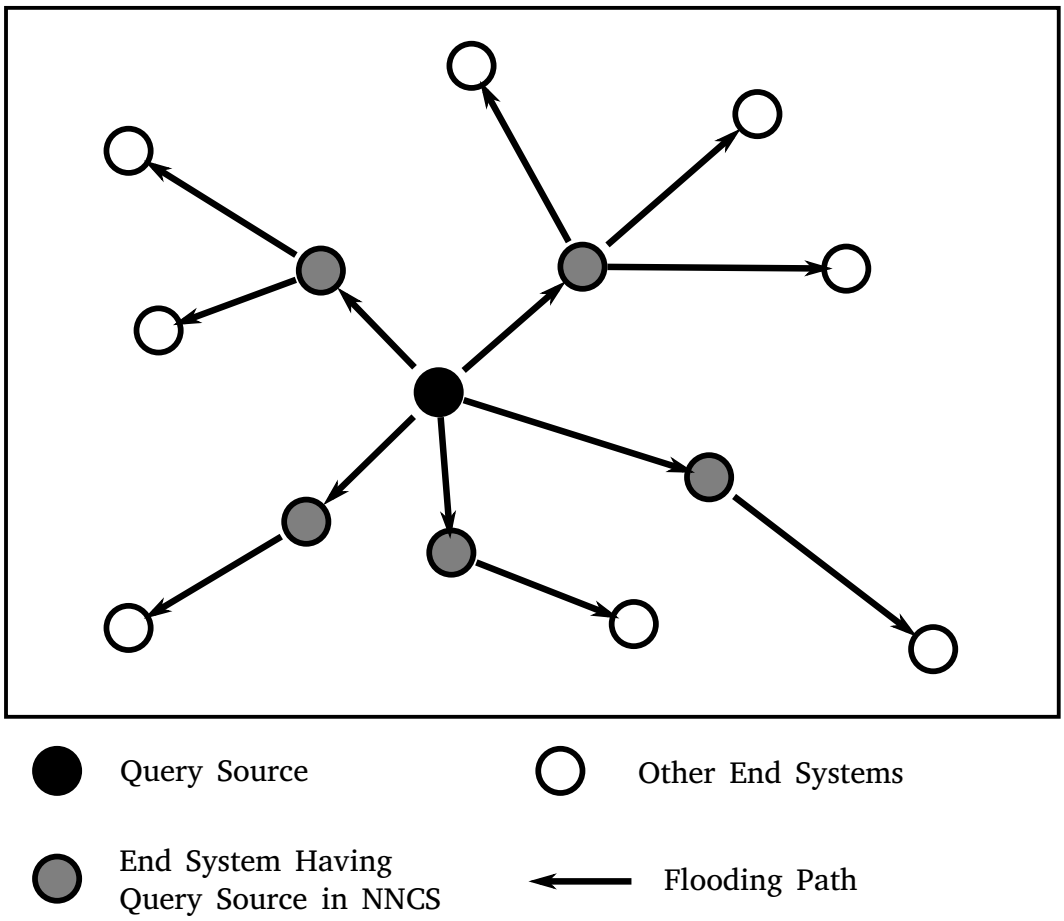

Figure 7.8: Example of Query Flooding on Reverse Greedy Routing Paths

7.8 Solving Network Optimization Problems (QoS)

Providing quality of service (QoS) guarantees between two end systems A and B in an overlay network means that a path in the overlay network can be found on which a given set of parameters are satisfied. Frequently used QoS parameters are bandwidth, jitter, delay, etc. In the worst case, finding such paths in an overlay network is NP-Complete, since every possible path between A and B must be examined. In this Section, we propose a solution that considerably reduces the number of end systems examined when searching for the optimal QoS path.

7.8.1 Spatially Bounded Flooding

In order to find a path fulfilling the QoS requirements Q_r between two end systems A and B with positions $\mathcal{C}_A, \mathcal{C}_B$, we have to examine almost every possible path between A

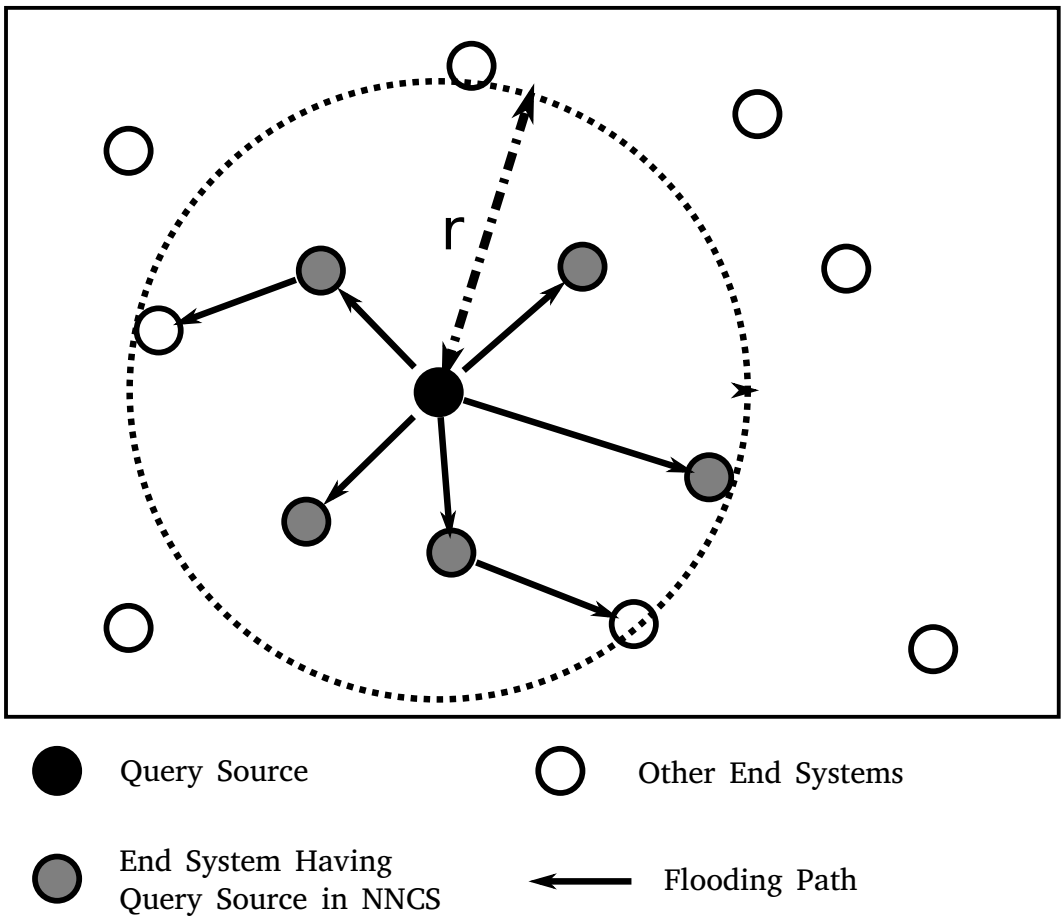

Figure 7.9: Example of Spacially Bounded Query Flooding on Reverse Greedy Routing Paths

and B. It is relatively easy to find a path which fulfills one of the QoS requirements, e.g. RTT or number of hops. However, it is rather unlikely for that path to fulfill also all other QoS requirements, e.g. bandwidth or jitter. Thus, in our search, we may have to examine all paths to ensure we find the one that best fulfills all required QoS parameters. In other words, to find a path which fulfills Q_r, we may have to flood the entire graph representing the overlay network.

Since we have an embedding of all end systems into a virtual space, we can reduce the extent of this flooding. To achieve this, we only consider the portion of the virtual space which fulfills the requirement concerning the maximal RTT (Q_{RTT}). In Figure 7.10, we depict such a portion in a two-dimensional space. As we can see, as long as the required RTT is larger than the distance between A and B in the virtual space, the portion of the two-dimensional virtual space fulfilling Q_{RTT} is an ellipse. Should the required RTT be smaller than the distance between the two end systems, the search can be terminated immediately, as it is physically impossible to find such a solution. In higher dimensions, this portion

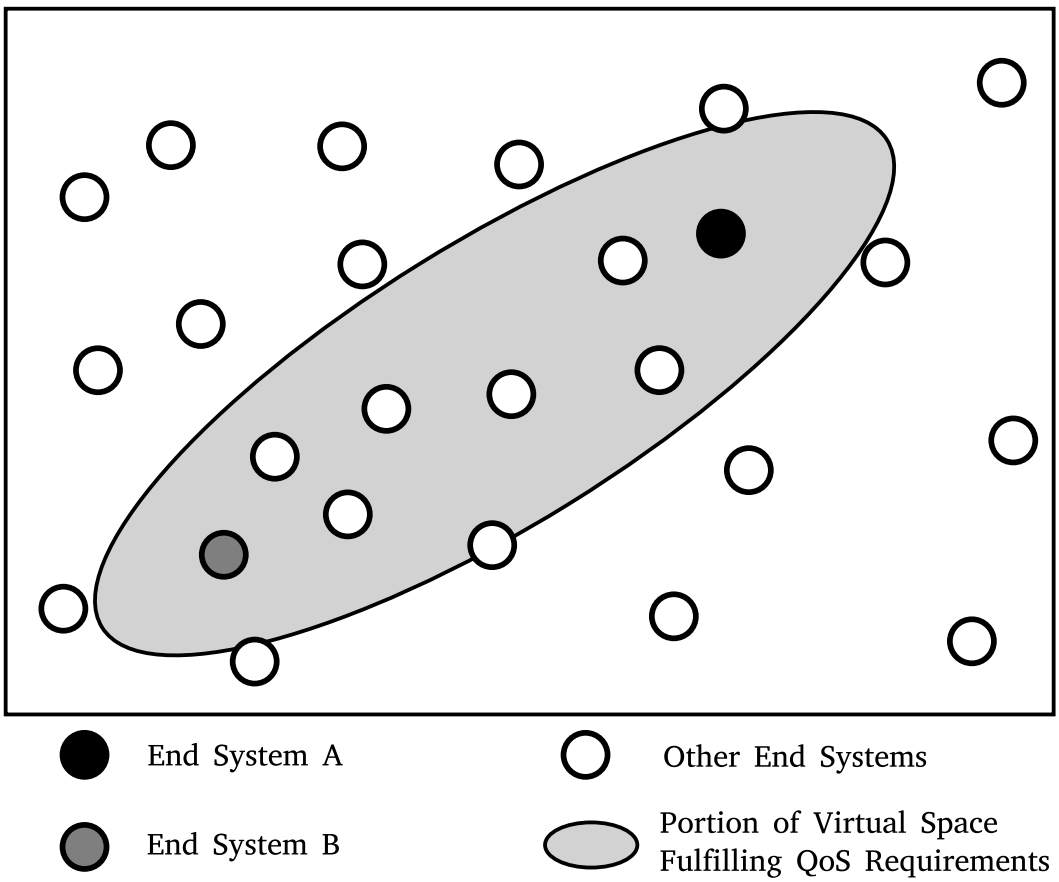

Figure 7.10: Portion of Virtual Space Covered by Q_{RTT}

would be an ellipsoid. The higher the required RTT, the larger (in terms of volume) this ellipsoid would be.

To find a path which fulfills all given QoS requirements Q_r, amongst which there must be a condition Q_{RTT} limiting RTT, we propose using the same NNCS overlay network as in Section 7.4. In analogy to the process used in the previous section, we propose to flood a spatially bound section of the overlay network. Unlike last time, however, this time we only flood the forward paths. In other words, this time, end systems do not need to compute the reverse paths and, hence, they do not need to gather the NNCS sets of their direct neighbors. We initialize the flooding with request messages for finding paths at the end system A. Each flood message M_f has following fields:

Q_r Original QoS request.
This field is used to identify the request to which M_f belongs. The request must state an upper limit for the RTT of the path (Q_{RTT}).

Q'_r Current QoS request.

This field contains the current values of the QoS parameters after the characteristics of the path so far have been removed. For example, at each "hop" in the overlay network, the RTT of the preceding "hop" is deducted from the RTT counter, resulting in the current Q'_{RTT}. The same applies for all other additive parameters of Q'_r such as jitter.

P Recorded path.
This field is a list of all end system addresses which the message passed along its path. This information is used to reconstruct successful paths.

In the beginning, an initial flood message M_f is sent out where the P contains only the address of the initiating end system A and Q'_r is equal to Q_r. End system a then calculates the distances for each of its neighbors $\{N_1 \dots N_c\}$. For each neighbor N_i, where the sum of the distances $d(A, N_i) + d(N_i, B)$ is lower than Q_{RTT}, a copy of M_f^i is sent. The message M_f^i contains a Q'_r which was adjusted with the characteristics of the logical link between a and N_i, e.g. RTT, jitter, etc.

Before an M_f^i is sent off, however, the availability of non-additive QoS parameters such as minimal bandwidth is also checked. If all parameters are fulfilled, e.g. the bandwidth between a and N_i is larger or equal to the requested bandwidth, the message M_f^i is sent to N_i. This procedure is repeated at each hop. All flood messages that reach end system b contain paths that fulfill the QoS requirements defined in Q_r. Depending on how reservations are performed, either end system b chooses one of the paths and sends reservation messages backwards through the path, or every path that fulfills Q_r is sent back to end system a, so that a can decide which path to reserve.

7.9 Evaluation

To evaluate our approach, we implemented the NNCS protocol using a event based network simulator [60]. Our network model delayed the delivery of messages by half the value of RTTs contained by the provided RTT matrix. We used RTT information data sets as our RTT matrix. One data set (PL) was provided by the Planet-Lab "all sites ping" experiment (for details see chapter 2.4.1) and contained 217 end systems. The other data set (KING) was obtained by measuring RTT distance between 462 end systems using the King method (for details see chapter 2.4.2). The overlay network was built using the Fisheye overlay (see chapter 5). To determine the positions of the end systems we used NetICE9 (see chapter 6 of this thesis). On top of NetICE9, we built an NNCS overlay network as described in 7.4.

All our results represent relative path stretch. The relative path stretch between two end systems is obtained by dividing the actual RTT on the path through the overlay network by the RTT along the optimal path in the underlying network. The relative path stretch can never be smaller than 1.0 as that is the value of the optimal path. The higher the value of the relative path stretch, the longer the path is relative to the optimal path. A value of 2.0, for example, means that the path through the overlay network was twice as long than the optimally achievable path.

In general, every overlay network can be only sub-optimal regarding the path stretch. The results of this evaluation can be used to estimate the relative error of found paths. We compared the performance of our approach (NNCS) to the performance of Pastry [7] in terms of the relative path stretch. We related the path streches found when using greedy unicast routing in the NNCS overlay network with the path stretches achieved by Pastry unicast routing. Our goal was to determine if the NNCS overlay protocol finds better paths in terms of path stretch than another well-known and established topology-aware overlay protocol. Another goal was to deliver estimates on the possible errors for the approaches described in 7.7 and 7.8.

Both Pastry and NNCS use random numbers for different aspects of the protocol. To reduce the influence of randomness on the results, we performed our simulations on the same data using different initial seeds. We performed every simulation with 20 different seeds, which were used to initialize pseudo random generators in the simulations. We collected all results and represent them as one cummulative distribution function (CDF) curve in Figures 7.11 and 7.12.

Since the number of dimensions used influences both, the precision of the RTT embedding performed by NetICE9 and the average number of neighbors in NNCS, we simulated NNCS with a varying number of dimensions. As stated in the related work [15, 16], the optimal number of dimensions for embedding distances obtained from the internet is somewhere in the range $5 - 7$. For this reason, we varied the number of dimensions used in our simulation from 3 to 6.

As Figure 7.11 shows, NNCS outperforms Pastry in the Planet-Lab data set already with $k = 3$ dimensions. With an increasing number of dimensions, the performance of NNCS improves further. The largest performance increase is achieved using 5 dimensions. With k larger than 5, there is no significant performance gain.

As Figure 7.12 shows, the situation with the King data set is similar. Here, however, Pastry performs better than NNCS in a 3-dimensional virtual space. With an increasing number of dimensions, the performance of NNCS improves. Similar as with the Planet-Lab data, the performance of NNCS in $k = 5$ and $k = 6$ dimensions does not significantly differ. Our theory is that with $k = 5$ dimensions, the maximal dimensionality of the provided data is almost achieved; thus, every increase in the number of dimensions does not further improve the embedding.

7.10 Conclusion

In this chapter, we have introduced a novel method for building an overlay network with end systems that are embedded into a virtual metric space: the nearest neighbors convex set (NNCS) overlay network protocol. The central advantage of NNCS is that greedy routing is always successful.

Based on NNCS, we have described a trivial unicast (greedy routing) and a multicast (flooding on reverse unicast greedy routing paths) protocol. We also described how the multicast protocol can be spatially bounded to achieve limited flooding that can be used for

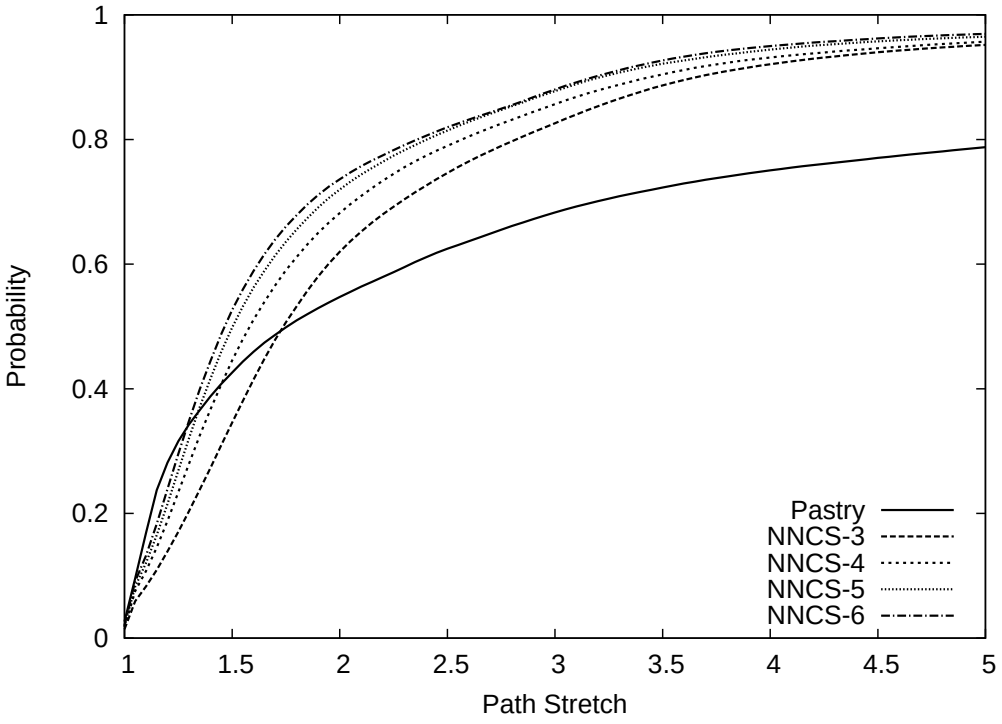

Figure 7.11: CDF of Relative Path Stretch in Simulations Performed Using Planet Lab Data

localized searches in the overlay network. A localized search can be used, for example, to implement a service discovery protocol. We also showed how NNCS can be used to reduce the overhead when searching for paths that fulfill given QoS requirements.

In our evaluation, finally, we showed that paths taken through an NNCS overlay network are more efficient than paths taken in Pastry.

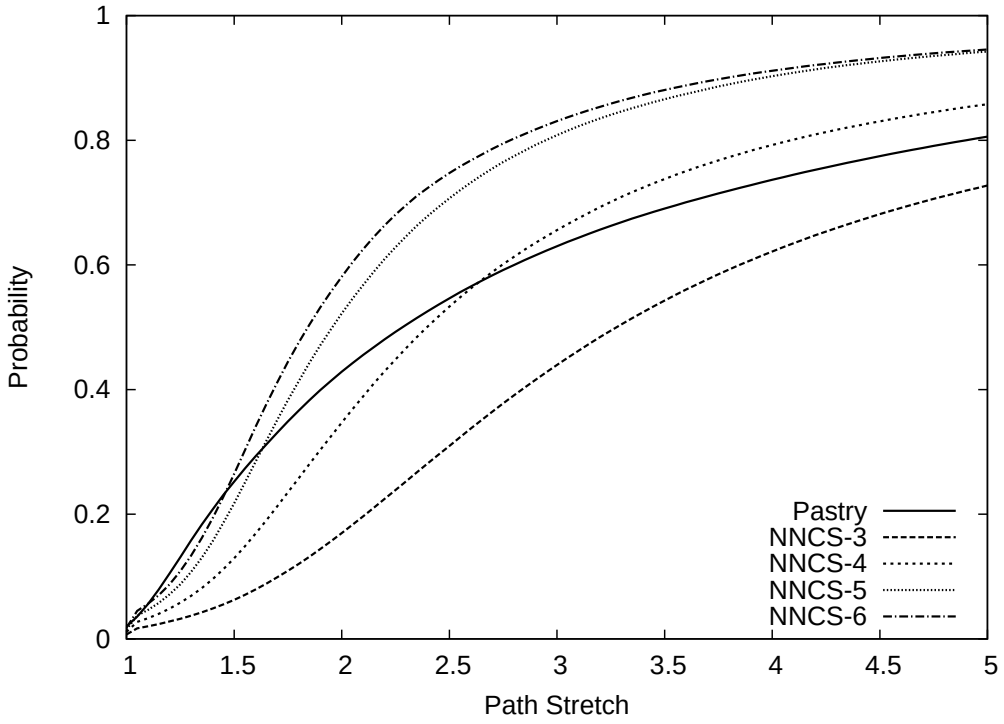

Figure 7.12: CDF of Relative Path Stretch in Simulations Performed Using Data Obtained With King Method

Chapter 8

Conclusion

8.1 Summary

The focus of this dissertation was on how to construct topology-aware overlay networks and on using them to provide new services such as topology-aware unicast and multicast routing, service discovery and finding QoS paths.

In chapter 3, we identified the computational overhead as one of the shortcomings of GNP. The reason of this computational overhead is the numerical approach that was chosen to minimize the function that, based on an input of RTTs measurements, determines the positions of the landmarks. The more parameters are used in the minimized function, the more the overhead of this function minimization increases. This increase is cubical relative to v, i.e. to the amount of parameters used in the minimized function. When landmarks are positioned in GNP, there are two factors which influence v: The first factor is how many landmarks are used (l); the second factor is the number of dimensions of the virtual space (k). It can be shown that v is determined as follows: $v = l \cdot k$. We further discussed how the number of dimensions k influences the error of the embedding and, consequently, influences the overall precision of the GNP algorithm: The higher the value of k, the smaller becomes the embedding error. We have shown that the computational overhead for positioning landmarks increases cubically with k. For that reason, we cannot just assign k to have the maximal value, since the computational overhead would become unmanageable. Next, we introduced a heuristic to determine the optimal value of k in order to achieve a balance between computational overhead and the overall error of the embedding. Finally, we proposed an algorithm to determine a good starting point for the numerical function minimization to further reduce the number of steps needed to compute the solution.

In chapter 4, we identified another shortcoming of GNP and similar approaches: The method used to determine the positions of the end systems in a virtual space does not always deliver the optimal solution. We identified the reason for this behavior in the way the positions are determined. Similar to the positions of the landmarks, the positions of end systems are determined by performing a function minimization. The function that is minimized (objective function) quantifies the error of the embedding for a given position by comparing the distances to landmarks in the virtual space with distances (RTTs) measured in the Internet.

The function minimization approach tries to find the position in the virtual space, where this error value is minimal. The problem of this approach is that the used objective function has multiple local minima. Every numerical function minimization starts at some (usually random) point of the parameter space and tries to iteratively "refine" the solution by finding a point nearby that has a lower function value. Using this approach, the numerical function minimization gets "stuck" in the first local minimum it finds. A local minimum is a point in a parameter space which is surrounded by points whose function value is higher than its own. If the function that is minimized has more than one local minimum, then function minimization will only find one of them. There is no guarantee that the found minimum is also the global minimum. Thus, in chapter 4, we first proved that the objective function used in GNP does indeed feature multiple local minima. Then, we proposed an algorithm that enumerates most of the local minima of the objective function and, hence, is able to find the optimal solution of the function minimization problem.

In chapter 5, we introduced a method of constructing a topology-aware overlay network, the fisheye overlay protocol. The fisheye overlay provides each participating end system with a fisheye view of the overlay network. In other words, each end system chooses its neighbors so that this choice contains not only a relatively detailed overview of its direct neighbors, but also includes some end systems which are further away. The fisheye view is assigned to each end system in a decentralized manner; every end system constructs its own fisheye view. In addition to this, the fisheye overlay protocol constructs an overlay network that is an undirected connected graph. These properties make the fisheye overlay a perfect overlay network for implementing topology-aware multicast services or an unstructured overlay network.

In chapter 6, we presented NetICE9, which is a new method for embedding end systems in virtual spaces. The scheme we proposed uses an approach similar to VIVALDI [16]. However, NetICE9 differs from VIVALDI in several aspects: It has smaller convergence times, displays improved stability and boasts better RTT predictions. This is achieved by using the fisheye overlay network presented in the chapter 5 to select the neighbors used to position each end system. NetICE9 also greatly reduces the problem of end systems oscillating in the virtual space. We achieved this by performing the error optimization towards all known neighbors at the same time as opposed to VIVALDI's approach where the error is optimized towards only one neighbor at a time.

Finally, in chapter 7, we presented a few new applications of positions of end systems in a virtual space. Most of these applications are based on the nearest neighbors convex set (NNCS) overlay networks we introduced in that chapter. NNCS is an overlay network that ensures the progress of greedy routing at each end system. It allows us to reach every end system with a known position through the overlay network by use of greedy routing exclusively. Greedy routing is very convenient since routing decisions are made at every end system independently of where the message was before and whether the end system has routed to that destination before. This eliminates the need to store any kind of routing information in the message or the end system. Also, there is no need for any kind of proactive or reactive mechanisms for finding routes in the overlay network. At the same time, since the positions of end systems are obtained by embedding RTTs measured in the

underlying network, greedy routing also optimizes the RTT stretch of paths taken through the overlay network. In the same chapter, we described how services such as unicast and multicast routing can take advantage of NNCS overlay networks and greedy routing. We also outlined how NNCS overlay networks can be used to implement localized service discovery with spatially bounded flooding. We then described a similar mechanism that can be used to optimize the discovery of QoS paths in NNCS overlay networks.

8.2 Outlook

The research presented in this dissertation is just a small contribution to the fields of RTT prediction in the Internet and construction of topology-aware overlay networks. There are still many open issues which need to be addressed. One of these issues is the question whether there is a better way to model RTT distances than using a Euclidean space. Shavitt et al. [31] show in their work that the Internet is better modeled by a Pointcaré disc rather than a k-dimensional Euclidean space.

It has been suggested that when it comes to RTT predictions, leaving the domain of metric spaces might yield even better results. For instance, one could consider the problem of RTT prediction as a data compression problem. However, if we did this, we would loose all the advantages we have gained when operating in metric spaces, such as the possibility to perform various geometry-based operations as described in chapter 7.

A hybrid approach can also be considered. For example, the one proposed by VIVALDI, where the position of every end system is modeled by its position in a virtual Euclidean space, and a non-metric component can be combined to reduce the embedding error. The major problem with such an approach concerns the semantics of this non-Euclidean component and how it should be incorporated into geometric approaches.

Bibliography

[1] D. Butskoy, "Traceroute for linux url: http://traceroute.sourceforge.net/," 2008.

[2] K. Lougheed and Y. Rekhter, "RFC 1267: Border Gateway Protocol 3 (BGP-3)," oct 1991.

[3] J. Moy, "OSPF specification," RFC 1131 (Proposed Standard), Internet Engineering Task Force, Oct. 1989, obsoleted by RFC 1247. [Online]. Available: http://www.ietf.org/rfc/rfc1131.txt

[4] ——, "OSPF Version 2," RFC 1247 (Draft Standard), Internet Engineering Task Force, July 1991, obsoleted by RFC 1583, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc1247.txt

[5] C. Hedrick, "Routing Information Protocol," RFC 1058 (Historic), Internet Engineering Task Force, June 1988, updated by RFCs 1388, 1723. [Online]. Available: http://www.ietf.org/rfc/rfc1058.txt

[6] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup protocol for internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, 2003.

[7] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware '01*. London, UK: Springer-Verlag, 2001, pp. 329–350.

[8] S. R. et al., "A scalable content-addressable network," in *Proceedings of ACM SIGCOMM*, 2001.

[9] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *GPS theory and practice : Second Edition*. Springer, May 1997.

[10] J. Crowcroft and J. P. Onions, "RFC 1165: Network time protocol (NTP) over the OSI remote operations service," June 1990, status: EXPERIMENTAL.

[11] S. Floyd and K. Fall, "Router mechanisms to support end-to-end congestion control," Lawrence Berkeley National Laboratory, Tech. Rep., 1997.

[12] A. Akella, J. Pang, B. Maggs, S. Seshan, and A. Shaikh, "A comparison of overlay routing and multihoming route control," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 93–106, 2004.

[13] S. Savage, T. A. A. Aggarawl, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan, "Detour: a case for informed internet routing and transport," *IEEE Micro*, vol. 19, pp. 50–59, 1999.

[14] Y. Shavitt and T. Tankel, "Big-bang simulation for embedding network distances in euclidean space," *IEEE/ACM Trans. Netw.*, vol. 12, no. 6, pp. 993–1006, 2004.

[15] T. S. E. Ng and H. Zhang, "Predicting internet network distance with coordiantes-based approaches," in *IEEE Infocom02*, New York / USA, June 23-27 2002.

[16] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: a decentralized network coordinate system," in *SIGCOMM '04*. New York, NY, USA: ACM Press, 2004, pp. 15–26.

[17] T. S. E. Ng and H. Zhang, "A network positioning system for the internet," in *USENIX 2004*, Boston MA, USA, June 2004, pp. 141–154.

[18] D. G. Andersen, N. Feamster, S. Bauer, and H. Balakrishnan, "Topology inference from bgp routing dynamics," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. New York, NY, USA: ACM, 2002, pp. 243–248.

[19] Z. M. Mao, J. Rexford, J. Wang, and R. H. Katz, "Towards an accurate as-level traceroute tool," in *SIGCOMM '03*. New York, NY, USA: ACM, 2003, pp. 365–378.

[20] J. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, vol. 7, pp. 308–313, 1965.

[21] S. Lee, Z.-L. Zhang, S. Sahu, and D. Saha, "On suitability of euclidean embedding of internet hosts," in *SIGMETRICS '06/Performance '06: Proceedings of the joint international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2006, pp. 157–168.

[22] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, "Measurement based analysis, modeling, and synthesis of the internet delay space," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 85–98.

[23] E. K. Lua, T. Griffin, M. Pias, H. Zheng, and J. Crowcroft, "On the accuracy of embeddings for internet coordinate systems," in *In IMC*, 2005.

[24] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A global internet host distance estimation service," *IEEE/ACM Transactions on Networking*, vol. 9, no. 5, pp. 525–540, October 2001.

[25] C. Pelsser, "Using virtual coordinates in the establishment of inter-domain lsps," in *CoNEXT '05: Proceedings of the 2005 ACM conference on Emerging network experiment and technology*. New York, NY, USA: ACM, 2005, pp. 274–275.

[26] A. Dufour and L. Trajković, "Improving gnutella network performance using synthetic coordinates," in *QShine '06: Proceedings of the 3rd international conference on Quality of service in heterogeneous wired/wireless networks*. New York, NY, USA: ACM, 2006, p. 31.

[27] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson, "Design and implementation trade-offs for wide-area resource discovery," *ACM Trans. Internet Technol.*, vol. 8, no. 4, pp. 1–44, 2008.

[28] H. Lim, J. C. Hou, and C.-H. Choi, "Constructing internet coordinate system based on delay measurement," in *Internet Measurement Confgerence 03*, October 2003.

[29] M. Costa, M. Castro, A. Rowstron, and P. Key, "Pic: Practical internet coordinates for distance estimation," in *International Conference on Distributed Systems*, Tokyo, Japan, March 2004.

[30] M. Pias, J. Crowcroft, S. R. Wilbur, T. L. Harris, and S. N. Bhatti, "Lighthouses for scalable distributed location," in *IPTPS*, 2003, pp. 278–291.

[31] Y. Shavitt and T. Tankel, "On the curvature of the internet and its usage for overlay constructi on and distance estimation," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Com puter and Communications Societies*, Hong Kong / PRC, March 7-11 2004, pp. 374–384.

[32] Y. Mao, L. K. Saul, and J. M. Smith, "Ides: An internet distance estimation service for large networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2273–2284, 2006.

[33] B. Wong, A. Slivkins, and E. G. Sirer, "Meridian: a lightweight network location service without virtual coordinates," in *SIGCOMM '05*. New York, NY, USA: ACM, 2005, pp. 85–96.

[34] S. Ratnasamy, M. H, R. Karp, and S. Shenker, "Topologically-aware overlay construction and server selection," 2002.

[35] C. G. Plaxton, R. Rajaraman, and A. W. Richa, "Accessing nearby copies of replicated objects in a distributed environ ment," in *SPAA'97: Proceedings of the ninth annual ACM symposium on Parallel algorithms and architectures*. New York, NY, USA: ACM Press, 1997, pp. 311–320.

[36] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "Scribe: The design of a large-scale event notification infrastructure," in *Networked Group Communication, Third International COST264 Workshop (NGC'2001)*, ser. Lecture Notes in Computer Science, J. Crowcroft and M. Hofmann, Eds., vol. 2233, Nov. 2001, pp. 30–43.

[37] S. Banerjee, B. Bhattacharjee, and K. Christopher, "Scalable application layer multicast," in *SIGCOMM02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, vol. 32, no. 4. New York,USA: ACM, 2002, pp. 205–217. [Online]. Available: http://portal.acm.org/citation.cfm?id=633045

[38] Y. hua Chu, S. G. Rao, and H. Zhang, "A case for end system multicast (keynote address)," *SIGMETRICS Perform. Eval. Rev.*, vol. 28, no. 1, pp. 1–12, 2000.

[39] S. E. Deering, "Multicast routing in internetworks and extended lans," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 55–64, 1988.

[40] R. McGeer, "Planetlab: a worldwide deployment infrastructure for the next generation of network services," CERN, Geneva, Tech. Rep., 2004, cERN, Geneva, 13 May 2004.

[41] C. Yoshikawa, "Planetlab all-sites-pings experiment url: http://ping.ececs.uc.edu/ping/," 2006.

[42] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *SIGCOMM Internet Measurement Workshop 2002*, 2002.

[43] C. de Launois, "A stable and distributed network coordinate systems," Université catholique de Louvain, Tech. Rep., December 2004.

[44] L. Tang and M. Crovella, "Virtual landmarks for the internet," in *Internet Measurement Confgerence 03*, October 2003.

[45] D. Milic and T. Braun, "Optimizing dimensionality and accelerating landmark positioning for coordinates based rtt predictions," in *BROADNETS*, 2007, pp. 631–640.

[46] D. M. Y. Sommerville, *An Introduction to the Geometry of n Dimensions*. New York: Dover Publications, 1958, p. 124.

[47] L. M. Blumenthal, *Theory and Applications of Distance Geometry*. Oxford: Clarendon Press, 1953.

[48] J. Hietaniemi. (2006, Feb.) Comprehensive perl archive network. [Online]. Available: http://www.cpan.org/

[49] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics*, vol. 1, pp. 80–83, 1945.

[50] J. W. Eaton. (2006) Octave. [Online]. Available: http://www.gnu.org/software/octave/

[51] F. James, "Minuit tutorial: Function minimization," in *CERN Computing and Data Processing School*, Pertisau, Austria, 1972.

[52] W. T. Vetterling, S. A. Teukolsky, W. H. Prea, and B. P. Flannery, *Numerical Recipes in C++, The Art of Scientific Computing*, 2nd ed. Cambridge: Cambridge University Press, 2002.

[53] R. P. Brent, *Algorithms for Minimization without Derivatives*. Englewood Cliffs, NJ: Prentice-Hall, 1973.

[54] D. Milic and T. Braun, "Fisheye: Topology aware choice of peers for overlay networks," in *The 34th IEEE Conference on Local Computer Networks (LCN)*, Zurich, Switzerland, October 20-23 2009.

[55] P. Kirk. (2003) Gnutella - a protocol for a revolution. [Online]. Available: http://rfc-gnutella.sourceforge.net/

[56] Napster webpage: http://napster.com/.

[57] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. 1st Internat. Workshop on Peer-to-Peer Systems (IPTPS'02)*. Springer, 2002, pp. 53–65.

[58] C.-C. Yang and L.-P. Tseng, "Fisheye zone routing protocol: A multi-level zone routing protocol for mobile ad hoc networks," *Comput. Commun.*, vol. 30, no. 2, pp. 261–268, 2007.

[59] P. Francis, "Yoid : Extending the multicast internet architecture," April 1999, white paper, http://www.aciri.org/yoid.

[60] "OMNET++ community site," Available: http://www.omnetpp.org, 2009.

[61] "Freepastry," Available: http://freepastry.rice.edu/, 2009.

[62] M. Heissenbüttel, "Routing and broadcasting in ad-hoc networks," PhD Dissertation, University of Bern, Institute of Computer Science and Applied Mathematics, June 2005.

[63] P. Bose and P. Morin, "Online routing in triangulations," in *ISAAC '99: Proceedings of the 10th International Symposium on Algorithms and Computation*. London, UK: Springer-Verlag, 1999, pp. 113–122.

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: Milic Dragan

Matrikelnummer: 00-103-341

Studiengang: Informatik

Bachelor ☐ Master ☐ Dissertation ☒

Titel der Arbeit: Error Resilient and Robust Overlay Networks

LeiterIn der Arbeit: Prof. Dr. Torsten Braun

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Bern, 03.05.2010

Ort/Datum

Unterschrift

Curriculum Vitae

Personal Details

Name	Dragan Milić
Date of Birth	January 18, 1975
Address	Sahlistrasse 5
	CH-3012 Bern, Switzerland
Hometown	Kladovo, Serbia
Nationality	Serbian

Education

2004 – 2010	Ph.D. student in Computer Science at the University of Bern, Switzerland
2004	Master of Science in Computer Science, University of Bern, Switzerland
1998 – 2004	Study of Computer Science at TU Berlin, University of Bremen and University of Bern
1989 – 1993	Mathematical Gymnasium, Belgrade